

Міністерство освіти та науки України  
Тернопільський національний технічний університет імені Івана Пулюя

Кафедра  
комп'ютерно-інтегрованих  
технологій

## МЕТОДИЧНІ ВКАЗІВКИ

до виконання лабораторних робіт

з дисципліни

***“Програмнування систем реального часу”***

Тернопіль  
2017

УДК 681.3  
М54

Укладач:

*Чихіра І.В.*, канд. техн. наук, доцент.

Рецензенти:

*Золотий Р.З.*, канд. техн. наук, доцент.

Методичні вказівки розглянуто й затверджено на засіданні методичного семінару кафедри комп'ютерно інтегрованих технологій Тернопільського національного технічного університету імені Івана Пулюя протокол № 8 від 22 березня 2017 р.

Схвалено та рекомендовано до друку науково-методичною комісією факультету прикладних інформаційних технологій та електроінженерії Тернопільського національного технічного університету імені Івана Пулюя протокол № 8 від 24 березня 2017 р.

Методичні вказівки для виконання лабораторних робіт з дисципліни М54 «Програмування систем реального часу» / Укладач : Чихіра І.В. – Тернопіль : Тернопільський національний технічний університет імені Івана Пулюя, 2017 – 92 с.

УДК 681.3

Методичні вказівки до виконання лабораторних робіт з дисципліни “Програмування систем реального часу” (для студентів напрямку підготовки 6.050202 Автоматизація та комп'ютерно-інтегровані технології).

© Чихіра І.В., ..... 2017  
© Тернопільський національний технічний університет імені Івана Пулюя, ..... 2017

# ЛАБОРАТОРНА РОБОТА № 1

## Тема: Основи роботи і налаштування ОСРЧ QNX

**Мета:** Ознайомлення з середовищем ОСРЧ QNX. Вивчення основних принципів початкового налаштування ОСРЧ QNX.

### 1. Поняття про ОСРЧ QNX

ОС жорсткого реального часу QNX Neutrino призначена в першу чергу для ринку вбудовуваних систем. Такі системи, як правило, працюють в автономному режимі і не потребують адміністрування.

Перш ніж почати говорити про операційну систему жорсткого реального часу QNX, потрібно спочатку визначитися, що взагалі означає "операційна система жорсткого реального часу"?

Отже, під системою реального часу розуміється така інформаційна система, у якій коректність вихідної інформації залежить не тільки від правильності застосованих алгоритмів, але і від часу появи результатів обробки інформації. Тобто при запізненні результатів вони можуть бути або марними, або збиток у результаті запізнення може бути нескінченно великий.

Цей часовий критерій умовно розділив операційні системи на два класи – операційні системи загального призначення (General Purpose Operation Systems – GPOS) і операційні системи реального часу (Real Time Operation Systems – RTOS).

Основна задача систем загального призначення – ефективний поділ ресурсів ЕОМ (таких як процесорний час, оперативна пам'ять і т.п.) між декількома одночасно виконуваними програмами. Такі операційні системи поставляються з багатим набором прикладних програм і мають розвинутий графічний інтерфейс, що дозволяє користувачам працювати з програмами, не задумуючись над внутрішніми механізмами ОС.

Операційні ж системи реального часу розробляються, припускаючи наявність зовнішніх джерел даних. Основна задача ОСРЧ – вчасно обробити запит, всі інші аспекти функціонування ЕОМ відходять на другий план. Тому ОСРЧ поставляються в комплекті з різноманітними засобами розробки додатків. Іншими словами, покупцями ОСРЧ є не кінцеві користувачі, а розроблювачі програмного забезпечення.

Часто говорять про операційні системи "м'якого" і "жорсткого" реального часу. ОС **жорсткого** реального часу ГАРАНТУЄ виконання якихось дій за визначений інтервал часу. А ОС **м'якого** реального часу ЯК ПРАВИЛО ВСТИГАЄ виконати задані дії за заданий час. Тобто ОС м'якого реального часу виконує задачу з якимсь значенням ймовірності.

### 2. Графічний інтерфейс користувача Photon

#### 2.1. Архітектура графічного середовища

Стандартним графічним середовищем в ОСРЧ QNX є графічний інтерфейс користувача Photon.

За аналогією з операційною системою QNX, графічне середовище Photon microGUI являє собою графічне мікроядро із сімейством процесів, що розширюють

функціональність графічного мікроядра (або "графічного сервера") та взаємодіють за допомогою стандартного QNX-механізму передачі повідомлень. Саме графічне мікроядро Photon являє собою невеликий процес, що реалізує лише декілька фундаментальних примітивів, що використовуються зовнішніми процесами для виконання різних задач користувацького інтерфейсу.

Сервер Photon не працює з вікнами, не взаємодіє з пристроями вводу (мишею, клавіатурою і т.ін.). За безпосереднє промальовування зображень відповідає графічний драйвер. Задача графічного драйвера – відображення інформації, що отримується від сервера шрифтів і від інтерпретатора графічного потоку. За введення інформації за допомогою миші, клавіатури й інших пристроїв відповідає драйвер вводу.

Варто помітити, що Photon успадковує мережеву прозорість ОС QNX — додаткові графічні драйвери можуть використовуватися для розширення графічного простору Photon за рахунок фізичних дисплеїв інших вузлів мережі. При цьому можна легко забезпечити дублювання зображення.

Photon використовує кодування Unicode, що забезпечує введення і вивід тексту, написаного на різних мовах.

Для того щоб забезпечити повнофункціональне графічне середовище, що дозволяє користувачам маніпулювати вікнами додатків, зміною їхніх розмірів, переміщенням, згортанням і т.ін., використовується *віконний менеджер*. Він також підтримує панель задач. Робочий стіл разом з меню швидкого запуску додатків реалізовані за допомогою процесу *адміністратора робочого столу*. Вигляд робочого столу зображений на рис. 1.

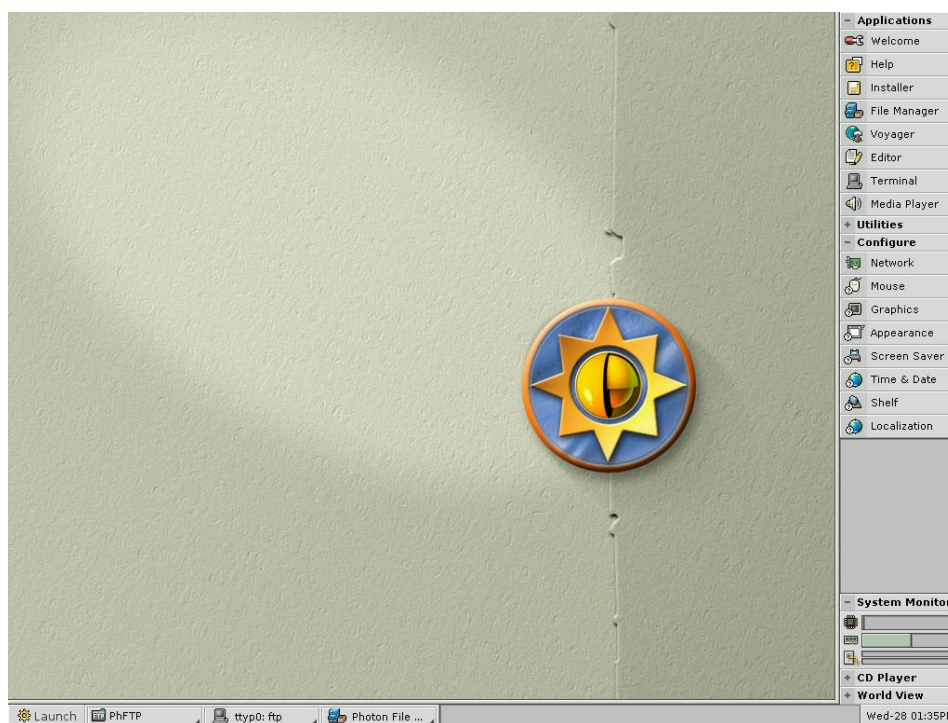


Рис. 1. Вигляд робочого столу

Таким чином, повна функціональність графічного інтерфейсу користувача QNX досягається набором процесів, що розширює можливості графічного сервера Photon.

## 2.2. Реалізація графічного середовища

У звичайній настільній системі Photon може запускатися одним із двох способів:

- вручну – з командного рядка в будь-який час після реєстрації в системі.
- автоматично – утилітою `tinit`, якщо не існує файлу `/etc/config/system/nophoton`.

У будь-якому випадку використовується командний сценарій `/usr/bin/ph`, що виконує запуск усіх необхідних компонентів графічного середовища в залежності від конфігурації системи. От що робить цей сценарій:

1. Запускає утиліту зондування графічного обладнання `crtrtrp`; якщо таке обладнання знайдене і розпізнане, то `crtrtrp` запускає програму `devgt-iographics` для визначення доступних графічних режимів відеокарти. Програма `devgt-iographics` записує результати своєї роботи у файл `/etc/system/config/graphics-modes`.

2. Запускається графічний сервер Photon. Якщо існує перемінна оточення `LOGNAME` (а це означає, що ви її або навмисно ініціалізували в командних скриптах, або ви вже пройшли реєстрацію за допомогою утиліти `login`), то Photon стартує відразу. У протилежному випадку Photon запустить утиліту `phlogin` для реєстрації користувача в системі. Можна заборонити користувачеві вихід з Photon у командний рядок, привласнивши перемінній `PHEXIT_DISABLE` значення 1.

3. Програма `crtrtrp` запускає адміністратор графічного виводу `io-graphics`, використовуючи командний рядок з файлу `/etc/system/config/graphics-modes`. Процес `io-graphics` завантажує інтерпретатор графічного потоку `gri-photon.so` і сервер шрифтів. По замовчуванню, `io-graphics` завантажує сервер шрифтів, реалізований у вигляді поділюваного об'єкта – `phfont.so`. Можна вказати адміністраторові графічного виводу запускати сервер шрифтів у вигляді окремого процесу `phfont`, створивши перемінну оточення `PHFONT__USE__EXTERNAL`. Окремий процес сервера шрифтів може знадобитися для того, щоб його можна було використовувати з інших вузлів мережі.

4. Запускається процес `inputtrap` для зондування пристроїв вводу. Він визначає, з якими аргументами необхідно запускати драйвер-адміністратор графічного введення `devi-hirun` і запускає його. Результат своєї роботи `inputtrap` зберігає у файлі `/etc/config/trap/input.im'я_вузла`.

5. Запускається утиліта `font sleuth`, що вказує серверові шрифтів `phfont`, у яких каталогах знаходяться шрифти.

6. Потім запускаються процеси `bkgdmgr` (малює фон робочого столу), `wmswitch` (дозволяє переключатися між відкритими вікнами додатків, використовуючи комбінацію клавіш "Alt"- "Tab"), `saver` ("зберігач екрану").

Крім того, сам сервер Photon запускає віконний менеджер `rwm` і адміністратор робочого столу `shelf` (рис. 2).

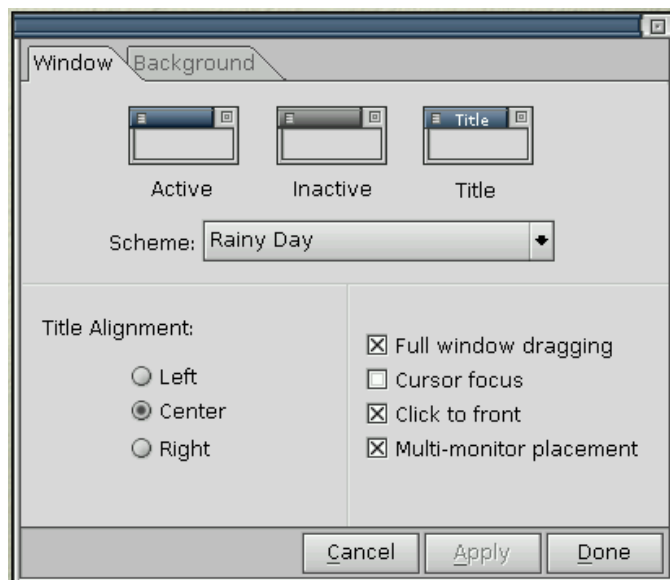


Рис. 2. Віконний менеджер

### 2.3. Утиліти конфігурування

Компоненти Photon можна конфігурувати за допомогою декількох утиліт. Для того, щоб змінити налаштування віконного менеджера, використовується утиліта `rwmopts` (рис. 3). Щоб її запустити потрібно в вікні терміналу набрати команду `rwmopts` або вибрати пункт з головного меню `Launch⇒Configure⇒Appearance` чи клацнути на піктограмі `Appearance` у меню швидкого запуску.

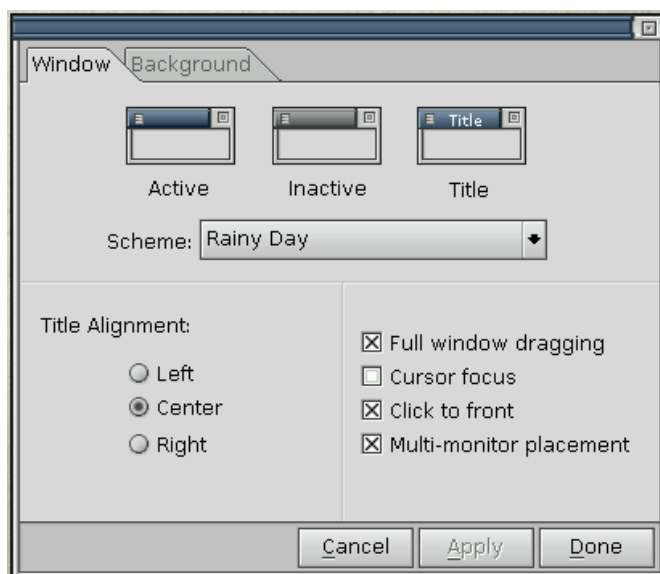


Рис. 3. Утиліта `rwmopts`

Вкладка `Background` утиліти `rwmopts` дозволяє налаштувати параметри адміністратора фону `bkgdmgr`.

Налаштування адміністратора графічного виводу `io-grafics` виконуються за допомогою програми `phgrafx` (рис. 4), яку можна запустити виконавши відповідну команду або вибравши її у головному меню чи в меню швидкого запуску.

У вікні цієї утиліти можна задати, який відеодрайвер варто використовувати, значення розширення, глибини кольору і частоти відновлення екрана.

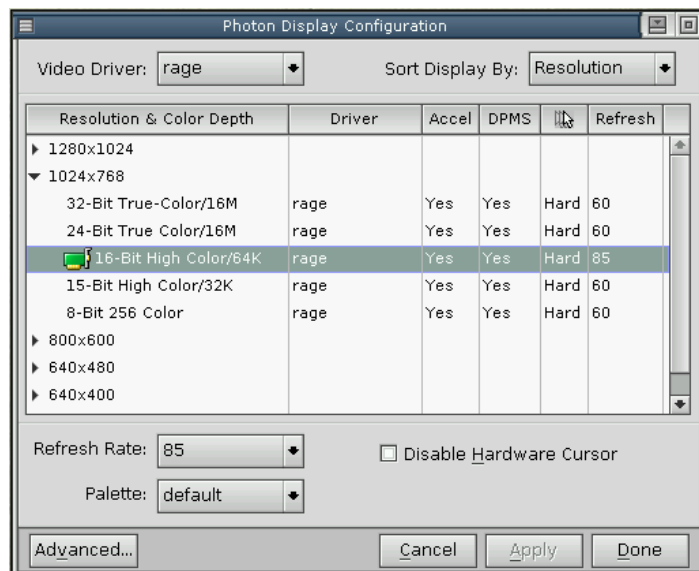


Рис. 4. Налаштування дисплею

Налаштування миші може виконуватися за допомогою утиліти input-cfg (Mouse) (рис. 5).

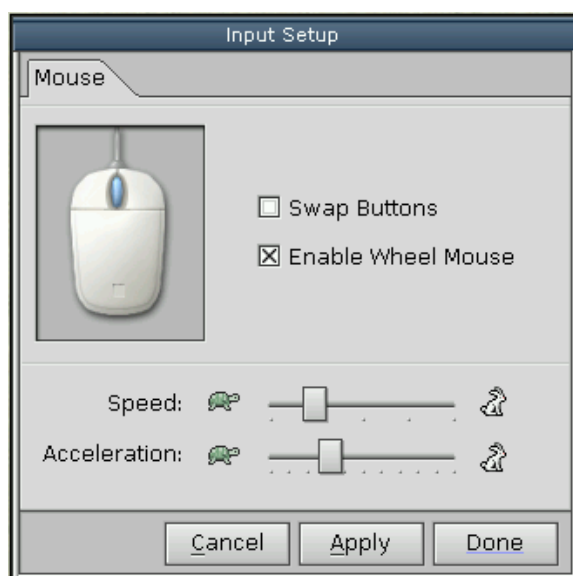


Рис. 5. Налаштування миші

Дозволити або заборонити віддалене з'єднання з локальною сесією Photon можна за допомогою утиліти phrelaycfg (рис. 6) (Launch⇒Configure⇒Remote Access).



Рис. 6. Налаштування віддаленого з'єднання

Встановити параметри зберігача екрану, час відсутності сигналів вводу, через який спрацює зберігач екрану, пароль зберігача можна за допомогою утиліти `savcfg` (рис. 7) (Screen Saver).

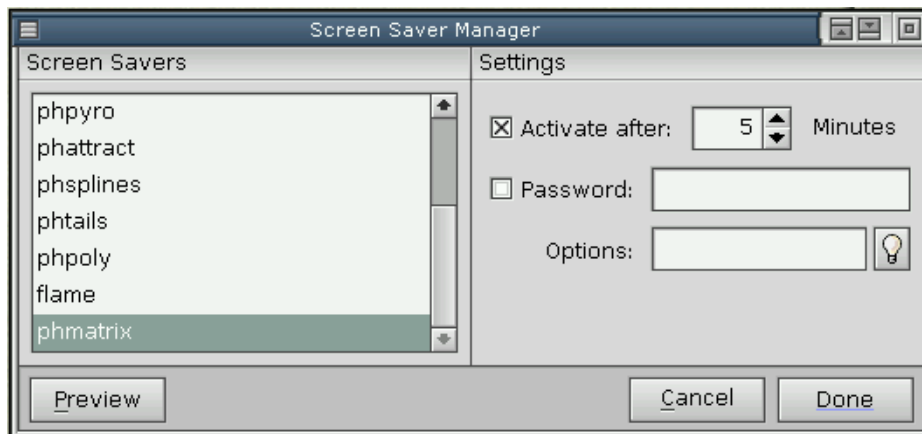


Рис. 7. Налаштування зберігача екрану

### 3. Налаштування QNX після інсталяції

Для нормальної роботи ОС після її інсталяції необхідно виконати декілька простих операцій. Якщо комп'ютер під'єднаний до мережі, то потрібно знати його IP-адресу і мережеву маску, IP-адреси шлюзу і сервера імен. Потрібно відмітити, що бажано задати хоча б ім'я хоста і його IP-адрес, навіть для одиничного комп'ютера, тому що багато програм розраховані для роботи в мережі і працюють коректно лише при наявності мережевих налаштувань (характерний приклад – QNX IDE).

Конфігурування TCP/IP здійснюється за допомогою команди `phlip` або ж вибравши піктограму Network в меню швидкого запуску чи в головному меню (рис. 8).



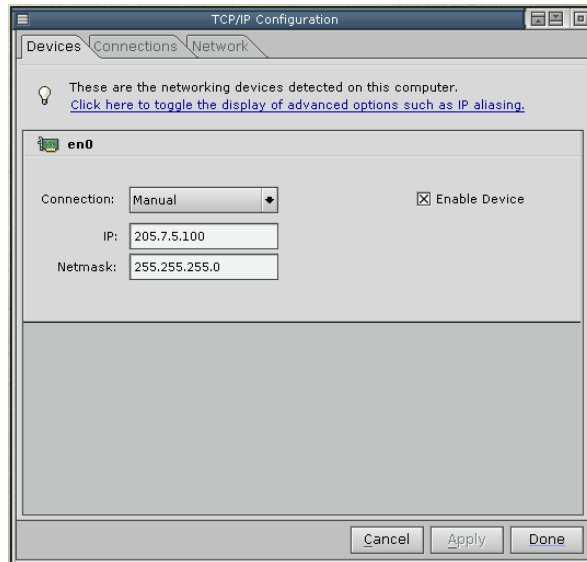


Рис. 8. Вікно налаштування мережі

Для вводу імені комп'ютера, адреси шлюзу і сервера імен потрібно перейти на вкладку Network (рис. 9).

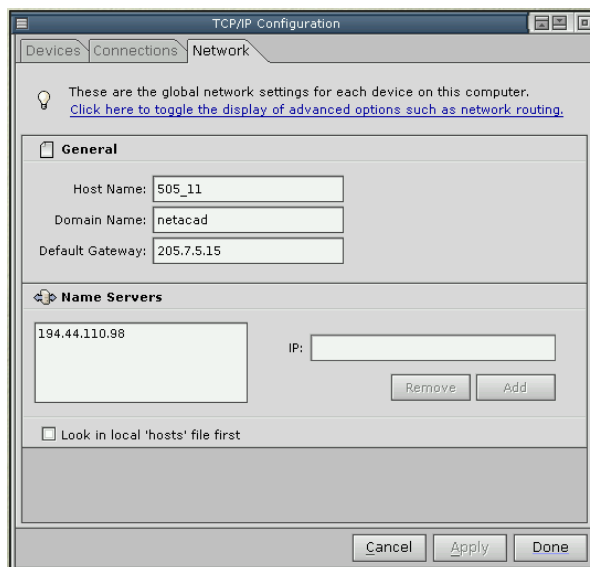


Рис. 9. Вікно налаштування мережі

Для налаштування локальних параметрів потрібно запустити програму User's Configuration. Для чого в меню робочого столу потрібно натиснути кнопку **Localization**. Тепер можна встановити часовий пояс (рис. 10).

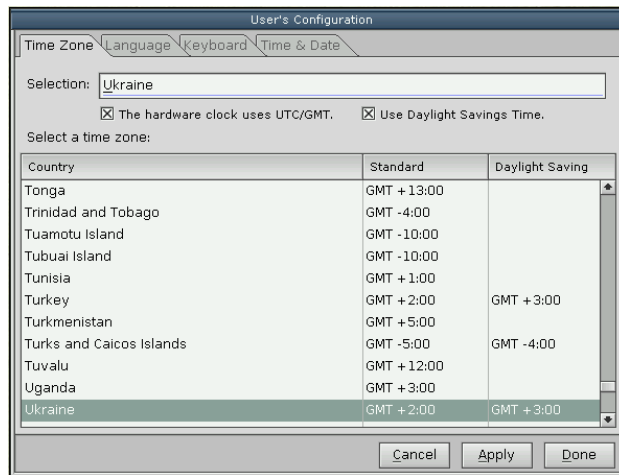


Рис. 10. Налаштування часового поясу

А також час і дату (рис. 11).

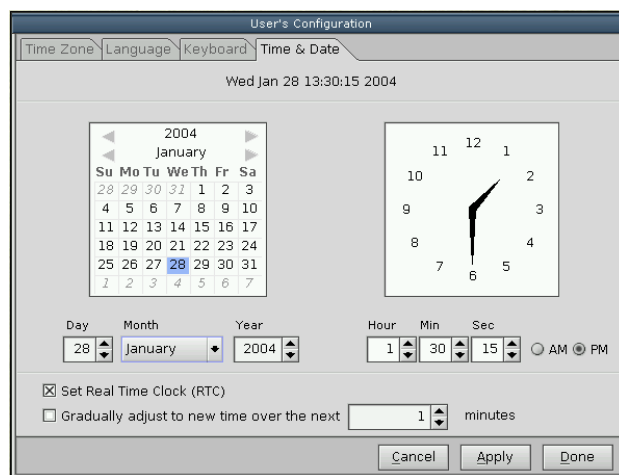


Рис. 11. Налаштування часу та дати

Вкладка Language призначена для встановлення мови. Проте, там немає ні української, ні російської мови. Російську мову можна встановити за допомогою вкладки Keyboard. Для цього потрібно перейти на цю вкладку, виділити рядок **Russian** і натиснути кнопку **Apply** (Застосувати зміни), а потім **Done** (Закрити вікно).

Програма User's Configuration вважає, що користувач використовує лише одну розкладку англійську чи російську. Для того, щоб встановити декілька мов, потрібно відредагувати файл /etc/system/trap/.KEYBOARD.myhost (зверніть увагу: myhost – це ім'я нашої машини) так, щоб він містив два рядки:

```
en_US_101.kbd
ru_RU_102.kbd
```

Зверніть увагу, що ім'я файлу починається з крапки ("."). Це означає, що файл схований, і засоби перегляду файлової системи, по замовчуванню, його не відображають. Щоб бачити сховані файли в файловому менеджері Photon, необхідно в меню Edit вибрати елемент Preferences. У вікні, що відкрилося, установити саме верхнє налаштування Show hidden files (Показувати сховані файли). Після цього не забудьте натиснути Apply.

Тепер при запуску додатків Photon можна буде вводити як російський, так і англійський текст (для переключення між розкладками клавіатури використовується комбінація лівих клавіш "Alt"-"Shift"). Однак у командному рядку ви не зможете ні вводити, ні читати російські букви. Вирішувати цю проблему покликаний пакет SWD Cyrillic Pack.

#### **4. Завдання на лабораторну роботу**

1. Запустити систему та зареєструватись.
2. Перейти в графічний режим роботи.
3. Здійснити настройку віконного менеджера за допомогою утиліти rwmports.
4. За допомогою утиліти io-grafics налаштувати адміністратор графічного виводу.
5. Налаштувати параметри миші.
6. Дозволити віддалене підключення до локальної сесії Photon.
7. Вибрати та налаштувати параметри зберігача екрану.
8. Здійснити налаштування мережевого інтерфейсу.
9. Налаштувати локальні параметри системи.
10. Завершити роботу системи.

#### **5. Порядок оформлення звіту по роботі**

1. На титульній сторінці друкарським шрифтом вказати назву університету, кафедри, назву і номер роботи, прізвище, ініціали, номер групи виконавця, прізвище і ініціали викладача, рік виконання роботи.
2. Вказати тему та мету роботи.
3. Привести опис дій при налаштуванні ОСРЧ QNX.
4. Зробити висновки.

## ЛАБОРАТОРНА РОБОТА № 2

**Тема:** Ознайомлення з інтегрованим середовищем розробки QNX та написання у ньому елементарної програми

**Мета:** Набуття елементарних навиків роботи в IDE QNX

### 1. Знайомство з платформою Eclipse

Компанія QSS, розробник QNX, традиційно не вела самостійних розробок інструментальних засобів для своїх ОСПЧ. Наприклад, у QNX4 штатним інструментом була система програмування C/C++ компанії Watcom (зараз ця торгова марка належить корпорації Sybase). Однак розвиток цього компілятора припинився, і в QNX6 була вбудована система програмування GNU CC, яка швидко розвивається і є досить популярною. Це рішення, поряд із широкою підтримкою специфікацій POSIX, стало важливим кроком у реалізації концепції "переносимості розроблювачів" – програміст, знайомий з будь-якою UNIX-подібною ОС, може приступати до розробки прикладних програм для QNX без перепідготовки.

Однак повноцінного інтегрованого середовища розробки (IDE) для QNX так і не було. Компанія Metrowerks (у даний момент належить корпорації Motorola) випустила IDE CodeWarrior для QNX, що працює в середовищі Microsoft Windows і дозволяє виконувати крос платформну розробку QNX-додатків для цільових систем на базі процесорів PowerPC і x86. CodeWarrior використовує компілятор GNU CC.

QSS постійно розширює перелік апаратних платформ для цільових QNX-систем і тому підтримки в IDE платформ PowerPC і x86 було явно недостатньо.

У листопаді 2001 року IBM передала вихідні тексти свого проекту Eclipse у вільне поширення на умовах CPL (Common Public License). Таким чином, утворився відкритий консорціум Eclipse.org (<http://www.eclipse.org>), який складався з компаній-виробників програмного інструментарію і був покликаний координувати подальший розвиток проекту. Метою створення консорціуму було об'єднання виробників різного програмного інструментарію в рамках єдиної технології інтеграції, що дозволяє створити універсальну розширювану багатоплатформну IDE. Компанія QSS також передала консорціуму Eclipse.org частину своїх розробок в області програмування на C/C++. У результаті, до складу пакету розроблювача QNX Momentics PE увійшла QNX IDE на базі Eclipse.

### 2. Поняття про QNX IDE

Для запуску QNX IDE потрібно вибрати в меню **Launch/Development** елемент **Integrated Development Environment**. Після цього відкриється вікно, що називається **робоче місце** (Workbench). Можна одночасно відкрити і використовувати декілька робочих місць.

При першому запуску QNX IDE створить у домашньому каталозі користувача папку з ім'ям *workspace*, із вмістом якої IDE і буде працювати. Цей вміст називається **ресурсами**. До ресурсів відносяться папки і файли, що складають проекти.

У вікні робочого місця можна відкривати вікна редакторів і представлень.

**Редактором** (editor) називається такий елемент (plug-in) IDE, що дозволяє переглядати і/або модифікувати ресурси. Наприклад, редактор C-файлів, редактор Make-файлів.

**Представленнями** (views) називаються такі елементи IDE, що дозволяють маніпулювати ресурсами, відображаючи (представляючи) їх у певному логічному вигляді. До представлень відносяться, наприклад, різні навігатори і таблиці властивостей елементів проектів.

Набір редакторів і представлень, оптимізований для виконання будь-якої спеціалізованої задачі, називається **перспективою** (perspective). При першому старті IDE відображається перспектива **Resource**. Крім неї, визначені наступні перспективи:

**C** – для проектів на мовах C/C++;

**Java і Java Type Hierarchy** – для Java-проектів;

**Debug** – для налагодження програм;

**Help** – для доступу до документації;

**Team** – для спільної роботи декількох програмістів з одним проектом;

**Plug-in Development** – для розробки нових елементів IDE;

**QNX Profiler** – для профілювання розроблених програм;

**QNX System Builder** – для формування вбудовуваних образів QNX і завантаження їх на цільові EOM;

**QNX System Analysis** – для візуалізації траси подій ядра, побудованої за допомогою пакета System Analysis Toolkit;

**QNX Information** – для моніторингу процесів, що виконуються на цільових системах.

На рис. 1 представлена перспектива **C/C++ Development**.

По суті справи, перспектива являє собою файл у XML-форматі, який описує редактори і представлення, що використовуються, а також їхнє розміщення у вікні Workbench. Тому, користувач може розмістити будь-які наявні редактори і представлення так, як йому це зручно, і оголосити отриману конфігурацію робочого місця як нову перспективу.

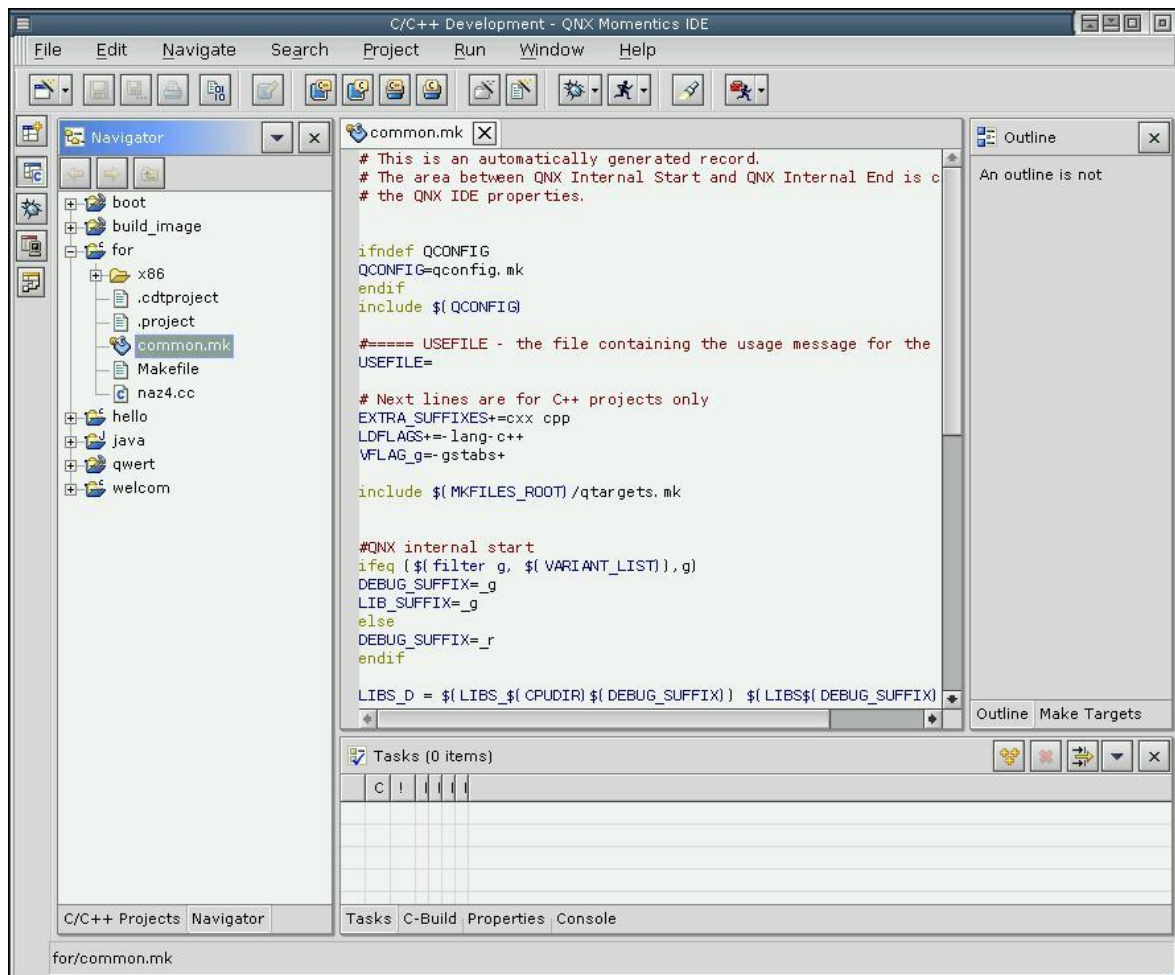


Рис. 1. Перспектива C/C++ Development

### 3. Конфігурування QNX IDE

Для робочого місця (Workbench) можна задати наступні настроювання:

**Perform build automatically on resource modification** – вказує, що необхідно виконувати збірку проекту при збереженні змін у будь-якому ресурсі проекту;

**Save all modified resources automatically prior to manual build** – зберігати змінені ресурси при запуску збірки проекту вручну;

**Link Navigator selection to active editor** – активізувати редактор ресурсу при виділенні імені ресурсу в представленні **Navigator**. І навпаки – при активізації редактора з відкритим ресурсом автоматично виділяти ім'я ресурсу в навігаторі (рис. 2).

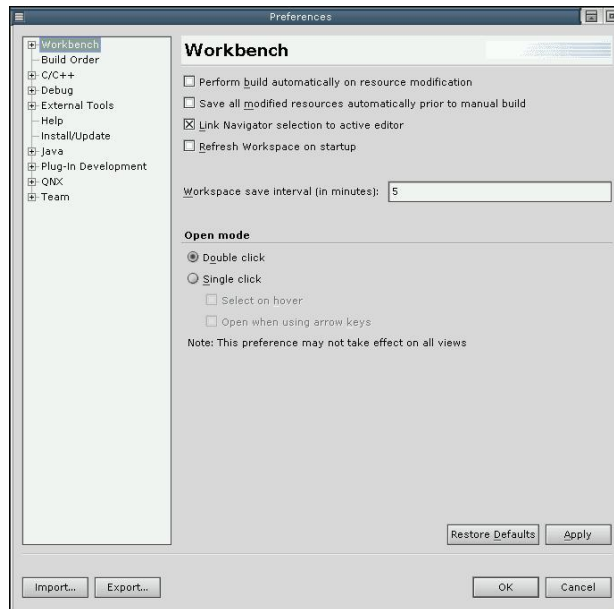


Рис. 2. Preferences

Для вікна **Workbench** можна задати, розміщення вкладки (вгорі чи внизу) для переключення між згрупованими редакторами і представленнями (рис. 3).

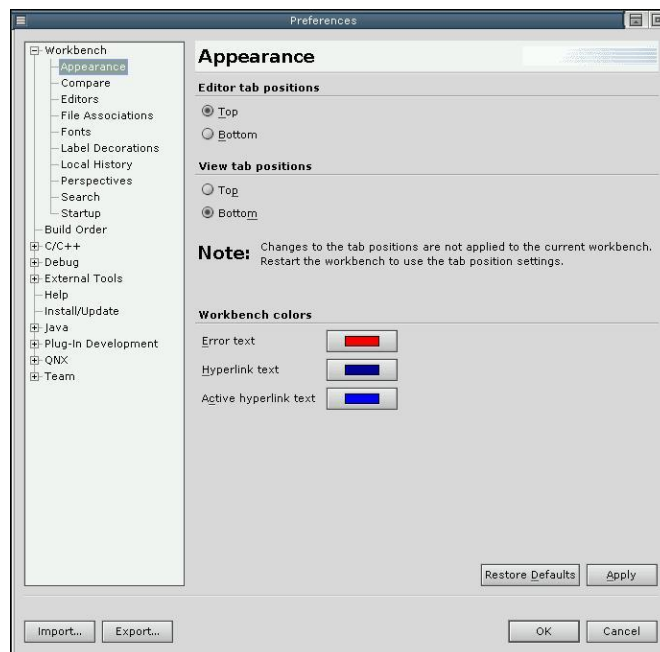


Рис. 3. Workbench

Для налаштування переглядів файлів, що порівнюються, використовується сторінка **Compare Viewers**:

**Synchronize scrolling between panes in compare/merge viewers** – установити загальну лінійку прокручування для порівнюваних або поєднуваних файлів;

**Show pseudo conflicts** – показувати так названі "псевдоконфлікти", тобто випадки, коли різні користувачі внесли однакові зміни у файл;

**Initially show ancestor pane** – показувати ресурс, що є загальним "предком" порівнюваних файлів (рис. 4).

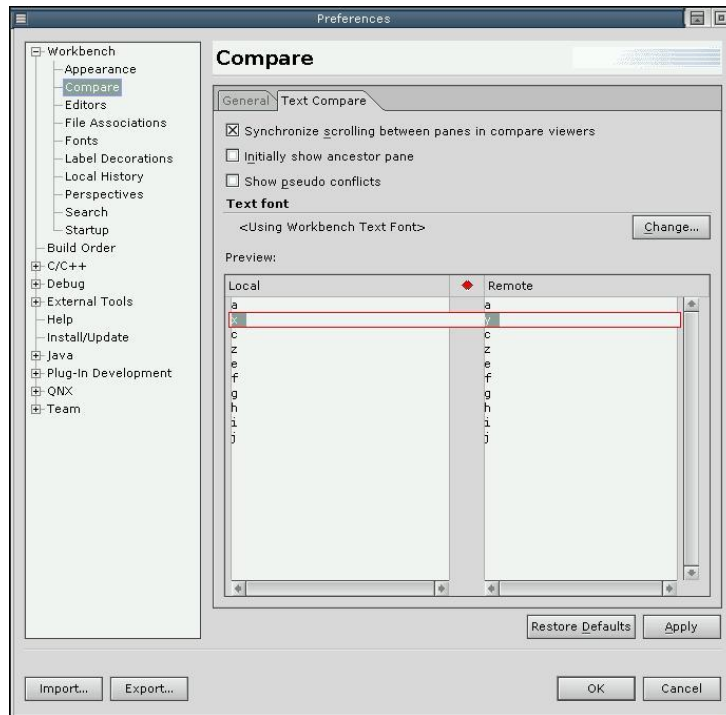


Рис. 4. CompareViewers

Редактори можуть бути вбудовані в IDE (plug-ins) або зовнішні (рис. 5).

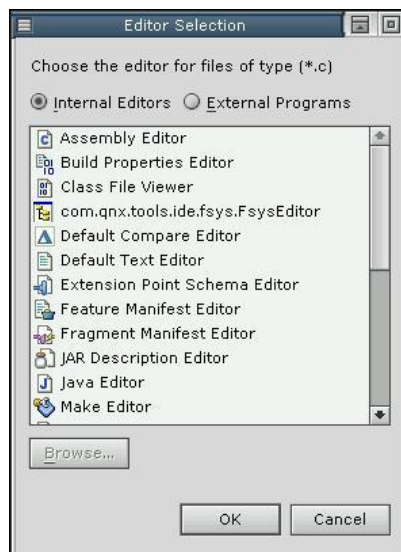


Рис. 5. Editor selection

У IDE передбачене налаштування основних шрифтів (рис. 6).



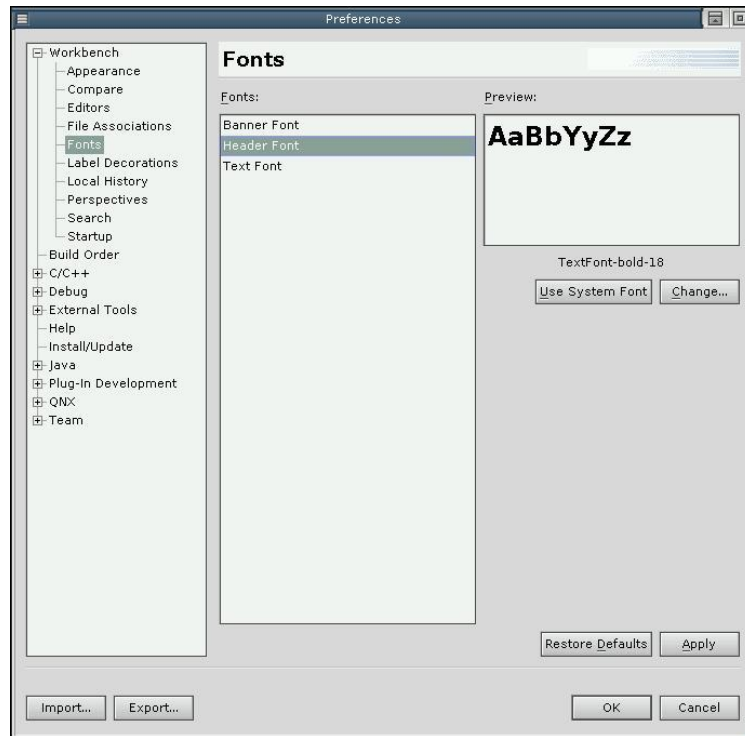


Рис. 6. Fonts

Для того щоб файли, що зберігають зміни ресурсів (локальну історію ресурсу), не займали великий дисковий простір, передбачені деякі обмеження: максимальна кількість днів збереження змін, максимальна кількість записів про зміну ресурсу, максимальний розмір файлу для збереження змін (рис. 7).

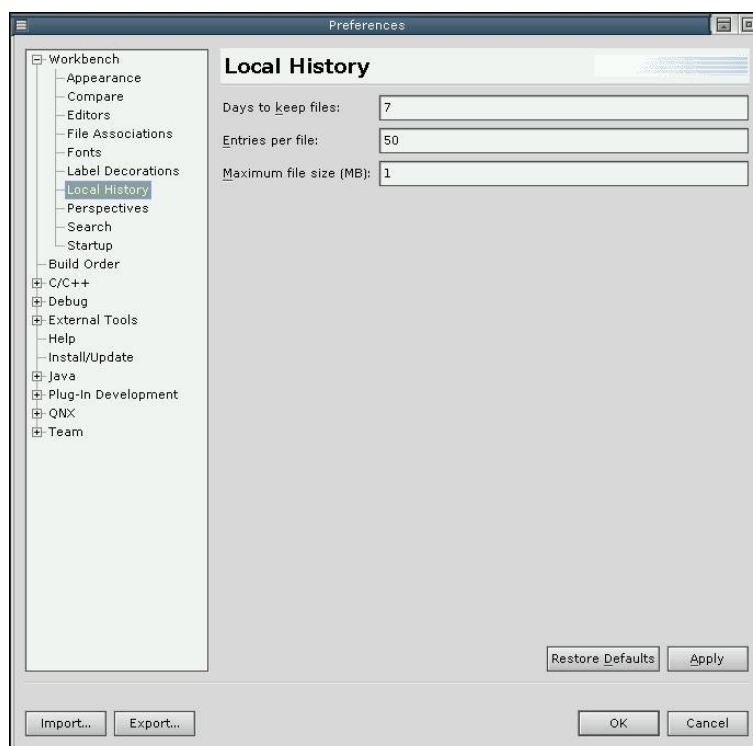


Рис. 7. Local history

У списку доступних перспектив можна вибрати перспективу, яка викликається по замовчуванню. На цій же сторінці вікна налаштувань робочого місця можна знищувати свої додаткові перспективи. Стандартні перспективи видалити не можна (рис. 8).

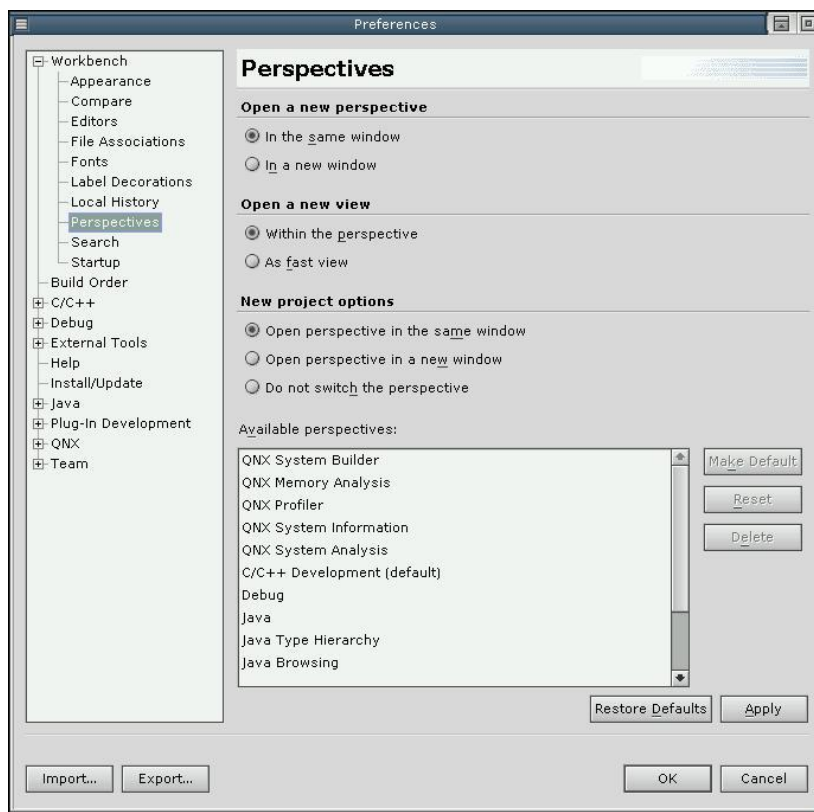


Рис. 8. Perspectives

Сторінка вікна налаштувань **Build Order** дозволяє встановлювати порядок зборки проектів. Ця властивість корисна, коли проекти залежать один від одного. Прапорець **Use Defaults**, встановлений по замовчуванню, відключає використання залежностей між проектами при їхній зборці.

#### 4. Приклад створення програми

Опишемо покроково створення програми для обчислення заданої функції –  $y=(x1+x2)^2$ , програма повинна робити запрошення на ввід значення  $x1$  та  $x2$  з клавіатури і вивести значення  $y$  на консоль.

Для цього необхідно виконати наступні дії:

1. Запускаємо IDE.

2. Вибираємо в меню **File⇒New⇒Project**, з'явиться вікно **New Project** зображене на рис. 9. У ньому зліва потрібно вибрати пункт C++, а справа пункт QNX C++ Application Project і тиснути на кнопці Next.

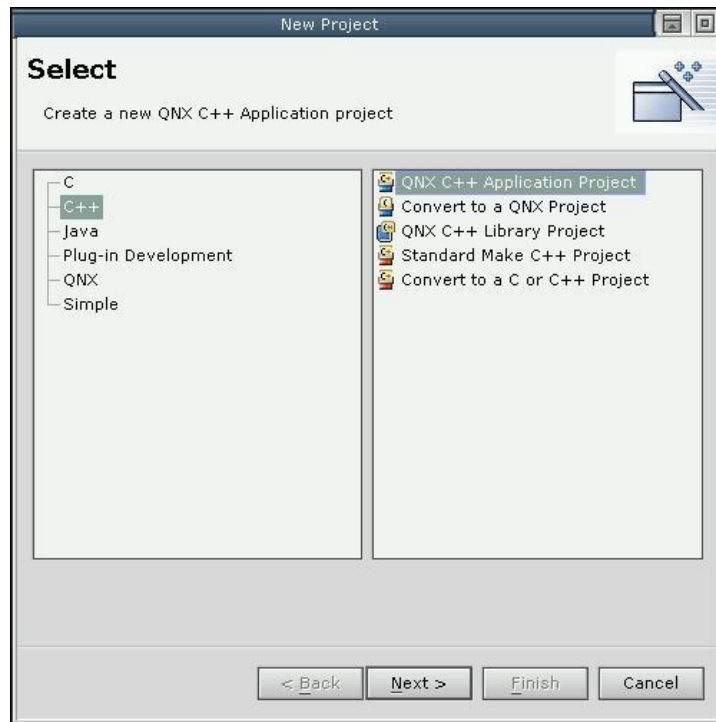


Рис. 9. New Project

3. У наступному вікні в полі **Name** потрібно вказати назву папки, де буде зберігатися проект. Ця назва буде і його ім'ям (наприклад, lab2).
4. Далі потрібно вибрати зі списку процесорну платформу, для якої буде побудований проект. Вибираємо для даного проекту платформу x86.
5. Для закінчення процесу створення проекту потрібно натиснути на кнопці **Finish**.
6. В результаті виконання попередніх дій, вигляд екрану буде наступним (рис. 10).

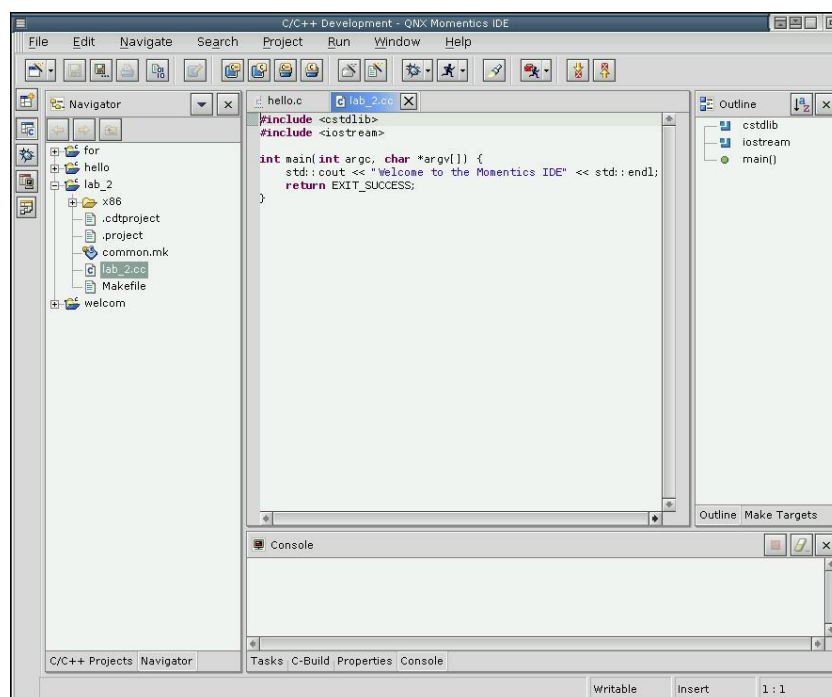


Рис. 10. C/C++ Development

Щоб програма виконувала потрібні операції, необхідно відредагувати її програмний код наступним чином:

```
#include <cstdlib>
#include <iostream>
#include <math.h>
int main(int argc, char *argv[])
{
    float x1, x2, y;
    std::cout << "Lab 2" << std::endl;
    std::cout << "Enter volume of x1" << std::endl;
    std::cin >> x1;
    std::cout << "Enter volume of x2" << std::endl;
    std::cin >> x2;
    y = powf((x1 + x2), 2);
    std::cout << "The resalt is:" << std::endl;
    std::cout << "y=" << y;
    return EXIT_SUCCESS;
}
```

7. Після внесених змін, їх потрібно зберегти. Для цього потрібно клацнути на кнопці **Save** в панелі інструментів або вибрати відповідний пункт з головного меню.
8. Далі для компіляції програми та запуску її на виконання потрібно в підменю **Run** вибрати пункт **Run...** Відкриється вікно **Launch Configuration**, яке використовується для створення, керування і виконання конфігурації програми. У списку зліва, потрібно вибрати тип конфігурації, яку потрібно створити, (вибираємо C/C++ Local) і тиснути на кнопці **New**. Далі в закладці **Main** в полі **Project**, за допомогою кнопки **Browse...**, потрібно задати розміщення проекту (виберемо папку lab2), а в полі **C/C++ Application** за допомогою кнопки **Search...**, вибрати файл запуску проекту (теж – lab2). Далі потрібно тиснути на кнопці **Apply**, а потім для виконання – **Run**.

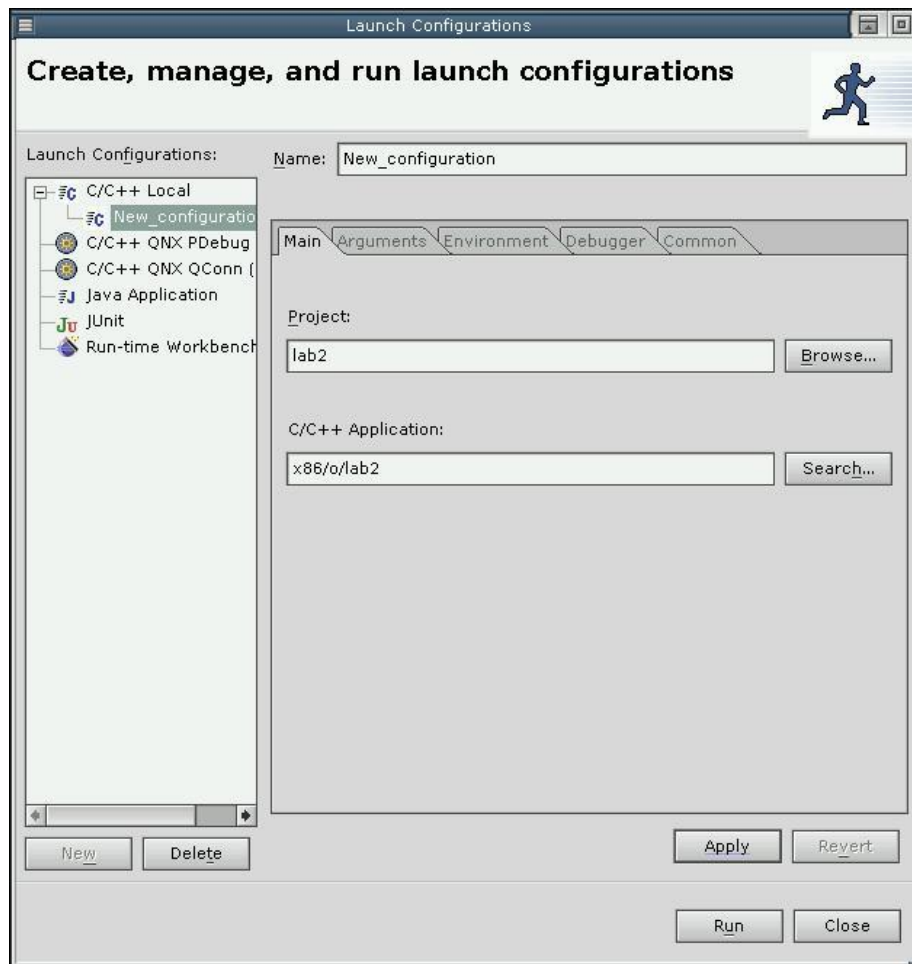


Рис. 11. Launch Configurations

## 5. Завдання на лабораторну роботу

1. Запустити систему та зареєструватись.
2. Перейти в графічний режим роботи.
3. Завантажити IDE QNX.
4. Написати програму обчислення арифметичного виразу у відповідності до варіанту.
5. Відкомпілювати та запустити програму на виконання.
6. Оформити звіт по лабораторній роботі.

Варіант	Завдання
1	$y = \frac{\sqrt{x_1^3 + x_2^5}}{1000} + 65$
2	$y = \frac{\sin^3(x_1) + 45 + x_2}{2x_1^4 + 4x_2}$
3	$y = 45 \sin(x_1 + x_2 + \lg_{10}(x_1 x_2^2))$
4	$y = \sqrt{56 + \frac{x_1 + x_2 + \sin(x_1 x_2)}{5 \cos(x_2^2)}}$
5	$y = \cos^3\left(\lg_{10} \frac{x_1 + 2x_2 + 9}{0.666}\right)$
6	$y = \frac{\cos^2\left(\lg_{10} \frac{x_2}{x_1}\right)}{45 + x_2}$
7	$y = \frac{4 \sin(3 + x_1 x_2)}{34 - 9x_2^3}$
8	$y = \cos(\sqrt{x_2} + 34x_1) - 4 \sin(x_2)$
9	$y = \sin(x_1 - x_2^3 + \sqrt{x_1}) - 1.3x_1^3$
10	$y = \operatorname{tg}(x_1 - x_2^2) + 31.55x_2x_1^2$
11	$y = 0.1x_1 \sin x_2 \cos x_1^4 + 55$
12	$y = 45x_1 \sin x_2 + \sqrt{9x_2x_1^3}$
13	$y = \sin^2\left(x_1 \frac{x_2}{x_1 + 53x_2^2}\right)$
14	$y = 23 \cos^2(x_1^3 x_2^5) + 2x_1$
15	$y = \sqrt{\frac{x_2^2 + x_1 / x_2}{16x_2x_1}}$

Щоб скористатися готовими математичними функціями, потрібно підключити бібліотеку `math.h`. Для цього потрібно на початку файлу в розділі опису стандартних заголовків функції `compute`, додати стрічку:

```
#include <math.h>.
```

У програмі можуть бути використані такі математичні функції:

`double cos( double )` – обчислює косинус кута;

`double sin( double )` – обчислює синус кута;

`double tan( double )` – обчислює тангенс кута;

`double log10( double )` – обчислює логарифм десятковий числа;

`double sqrt( double )` – обчислює корінь квадратний числа;

`double pow( double x, double y )` – здійснює піднесення до степеня у числа `x`.

## ЛАБОРАТОРНА РОБОТА № 3

### Тема: Побудова власного завантажувального образу QNX

**Мета:** ознайомлення з процесом завантаження ОС QNX. Набуття навиків створення власного завантажувального образу в IDE QNX.

#### 1. Початкове завантаження QNX Neutrino

Після включення живлення комп'ютера, на якому інстальована лише ОСРЧ QNX запускається процес початкового завантаження. Якщо ж в комп'ютері, поряд з ОС QNX, встановлено декілька інших ОС, то мультизавантажувач спочатку пропонує вибрати, яку ОС варто завантажувати. Після вибору розділу QNX починається процес початкового завантаження ОС.

Початкове завантаження QNX протікає поетапно:

1. Спочатку виконується код початкового завантажника (IPL – Initial Program Loader), що здійснює мінімальне конфігурування апаратури (конфігурує контролер пам'яті, системний годинник і т.п.), а потім виконує пошук на носії та завантаження в ОЗУ образу, який завантажує ОС. і передачу йому керування.

2. Потім виконується код загрузчика ОС (startup), що входить до складу завантажувального образу QNX. Загрузчик startup копіює і (якщо потрібно) розтискає завантажувальний образ QNX, визначає склад і конфігурацію апаратури, заповнює системну сторінку даних ОС і передає керування модулеві procnto, що також входить до складу завантажувального образу QNX.

3. Модуль procnto (мікроядро Neutrino + адміністратор процесів) запускає інші процеси, що входять до складу завантажувального образу QNX.

Варто відмітити, що в QNX звичайно використовуються два завантажувальних образи: основний образ міститься у файл /.boot, а резервний – у файл /.altboot. Вторинний загрузчик startup пропонує натиснути клавішу "ESC" для завантаження резервного образу.

Якщо нічого не натискати, то завантажуватися буде основний образ. Модуль procnto, одержавши керування, запускає послідовно всі процеси, що входять до складу завантажувального образу. Одним із процесів, що входять у стандартний образ, є diskboot. Основне призначення цього процесу – пошук базових образів файлової системи QNX на всіх доступних дисках і запуск командного сценарію /etc/system/sysinit.

Перш ніж приступити до роботи, diskboot пропонує натиснути "пробіл" для введення опцій завантаження:

Press the space bar to input boot options

Якщо нічого не натискати, завантаження продовжиться в автоматичному режимі.

Отже, якщо нічого не натискати, з'явиться повідомлення про сканування апаратури, потім, якщо diskboot знайшов більше одного розділу з файлами /.diskroot, то буде запропоновано вибрати, який з цих розділів буде кореневим. Останньою дією утиліти **diskboot** є виклик сценарію /etc/system/sysinit. Після цього стартує утиліта login або, якщо при інсталяції було зазначено запускати графічне

середовище Photon автоматично, утиліта phlogin. У будь-якому випадку знадобиться ввести ім'я і пароль.

## 2. Командні файли запуску

### 2.1. Командний сценарій *sysinit*

Як уже було сказано вище, процес diskboot перед своїм завершенням запускає командний сценарій `/etc/system/sysinit`. Задача цього сценарію – запустити процеси, що забезпечують необхідну функціональність ОС, він робить наступне:

- Запускає сервіс реєстрації системних подій (якщо він ще не запущений) `slogger`.
- Запускає адміністратор неіменованих каналів `pipe`.
- У випадку якщо це перший запуск системи після інсталяції, то запускає сценарій `/etc/rc.d/rc.setup-once`. Факт першого запуску встановлюється по відсутності файлу `/etc/system/package/packages`.
- Установлює часовий пояс. Інформація береться з файлу `/etc/TIMEZONE`.
- Запускає командний сценарій `/etc/rc.d/rc.rtc` (якщо такий існує) для налаштування годинника реального часу.
- Визначає ім'я ЕОМ. Інформація береться з файлу `/etc/HOSTNAME`.
- Запускає командний сценарій `/etc/rc.d/rc.devices` (якщо такий існує). Цей скрипт ініціює розпізнавання апаратних пристроїв.
- Якщо існує файл `/etc/system/config/useqnet` і запущений адміністратор мережного вводу/виводу `io-net`, то завантажується адміністратор мережного протоколу `Qnet`. Факт роботи `io-net` визначається по наявності префікса, який реєструється цим адміністратором, – каталог `/dev/ io-net`. Адміністратор протоколу `Qnet` реалізований у вигляді DLL, що розширює функціональність адміністратора `io-net`. Підключення виконується наступною командою:  
`mount -Tio-net nrm-qnet.so`
- Якщо існує файл `/.swapfile`, то він підключається в якості пристрою свопінгу.
- Запускає командний сценарій `/etc/rc.d/rc.sysinit` (якщо такий існує). Цей скрипт продовжує ініціалізацію системи.
- Якщо не вдається запустити `rc.sysinit`, то робиться спроба запустити командний інтерпретатор `Korn Shell` в інтерактивному режимі. Якщо стандартний інтерпретатор не може запуститися, робиться спроба запустити інтерпретатор з меншими вимогами до ресурсів – *Fat Embedded Shell* (`fesh`).

Таким чином, сценарій `sysinit` перед закінченням свого виконання викликає сценарій `rc.sysinit`.

### 2.2. Командний сценарій *rc.setup-once*

Цей командний сценарій викликається зі скрипта `/etc/system/sysinit` тільки один раз – при першому запуску системи.

Сценарій створює ряд каталогів: `/tmp`, `/var` з декількома підкаталогами, `/pkgs` з `/pkgs/repository`, `/root` (домашній каталог системного адміністратора). Файл для свопінгу `/.swapfile`, файл початкової конфігурації базової системи `/etc/system/package/packages`.



Якщо існує файл `/boot/setup.inf` (у цьому файлі зберігаються будь-які налаштування, виконані в процесі інсталяції), то запускається сценарій `/etc/rc.d/rc.setup-info`.

Потім перезапускається адміністратор пакетної файлової системи `fs-pkg` і генерується початкова пошукова база даних програми-переглядача електронної документації `helpviewer`.

Крім перерахованих дій, створюється робоча копія файлу `/etc/passwd`.

### **2.3. Командний сценарій `rc.devices`**

Командний файл `/etc/rc.d/rc.devices` викликається при кожному завантаженні QNX із сценарію `/etc/system/sysinit`.

Цей сценарій запускає адміністратор псевдотерміналів `devc-pty`, потім визначає каталоги, що містять інформацію про підтримувані пристрої (для локального вузла – `/etc/system/enum`). Після цього запускається адміністратор конфігурування апаратури `enum-devices`, який сканує підключені пристрої.

### **2.4. Командний сценарій `rc.sysinit`**

Командний сценарій `/etc/rc.d/rc.sysinit` викликається при кожному завантаженні QNX зі сценарію `/etc/system/sysinit` і призначений для виконання налаштувань, специфічних для даної ЕОМ, і запуску необхідних сервісів.

Цей сценарій запускає процес `dumper`, що зберігає "посмертні" `core`-файли процесів, що завершилися аварійно. Потім запускає сценарій `/etc/rc.d/rc.local`, якщо такий існує. Цей сценарій потрібний, якщо ви хочете додати свої команди ініціалізації, не редагуючи створені системою файли.

Остання дія сценарію `rc.sysinit` – запуск програми ініціалізації терміналу `tinit`. Ця програма запускає на терміналі утиліту входу в систему `login` або графічну оболонку `Photon` із графічною утилітою входу в систему `phlogin`. Який варіант використовувати, сценарій визначає по наявності або відсутності файлу `/etc/system/config/nophoton`.

Якщо запустити `tinit` не вдалося, сценарій намагається послідовно запустити командні інтерпретатори `ksh` (`sh` — це просто посилання на `ksh`) і `fesh`.

## **3. Побудова власного завантажувального образу**

Отже при завантаженні ОС спочатку виконується код початкового загрузчика IPL, потім викликається вторинний загрузчик `startup`, образ QNX, що завантажується (`/boot` або `/altboot`) поміщається в ОЗУ і, нарешті, керування одержує модуль `procnto`.

Варто зауважити, що IPL бувають "Warm-start" – для систем з BIOS (на момент старту IPL частина устаткування вже проініціалізована BIOS обчислювальної системи), і "Cold-start" – для систем без BIOS (всю ініціалізацію устаткування повинен виконувати сам IPL).

У QNX образи можуть бути двох типів – образ ОС (той що завантажується, тобто утримуючий `startup`-модуль, або що не завантажується) і образ вбудованої файлової системи (файлової системи `Flash`).

Образ ОС – це файл, що містить мікроядро, скомпоноване з адміністратором процесів, додатки і деякі дані, необхідні для функціонування цільової системи. Образ ОС можна розглядати як "маленьку файлову систему", тому що в образі є просте дерево каталогів, з якого `procnto` бере інформацію про файли, що утримуються в образі, їхнє місце розміщення в образі.

З погляду працюючої системи, образ ОС можна розглядати як файлову систему з доступом по читанню. Вміст образу можна подивитися за допомогою утиліти `dumpifs`.

Як образ ОС, так і образ файлової системи Flash можуть містити стиснуті дані – при створенні образу ці дані компресуються, при наступних операціях читання – декомпресуються автоматично. Слід зазначити, що використання механізму компресії/декомпресії сповільнює доступ до даних і створює додаткове навантаження на CPU цільової системи. Тому використовувати компресію рекомендується тільки для систем з дійсно обмеженим обсягом носія даних (Flash, ROM).

Образ ОС створюється програмою `mkifs` (MaKe Image FileSystem). Дані для своєї роботи ця утиліта бере з командного рядка і з файлу побудови — `buildflie`. Цей файл можна написати вручну, але зручніше скористатися QNX IDE. Для цієї мети в складі інтегрованого середовища розробки поставляється профіль інструментів (у термінах QNX IDE – "перспектива") – QNX System Builder.

Щоб зрозуміти, як створюється завантажувальний образ QNX для цільових систем, створимо простий образ ОС, що включає командний інтерпретатор Korn Shell і кілька утиліт – `ls`, `sin`, `pidin`.

Отже, запустимо IDE, створимо новий проект, указавши його тип майстрові створення проектів. При цьому майстер попросить ввести ім'я проекту (наприклад `build_image`) і визначити, для якої цільової системи будемо збирати образ (у даному випадку `Generic x86`, тому майстер за замовчуванням установить для майбутнього образу ім'я "generic").

Після того як усі попередні зведення зібрані, відкриється інструментальний профіль QNX System Builder (рис. 1).

Подивимося властивості образу, що визначені за замовчуванням, для цього клацнемо мишею на імені образу "generic" у вікні **Item Browser**.

Змінимо деякі властивості образу: оскільки у нас система з BIOS, то змінимо вміст полів **Boot file** і **Startup** редактора властивостей (рис. 2).

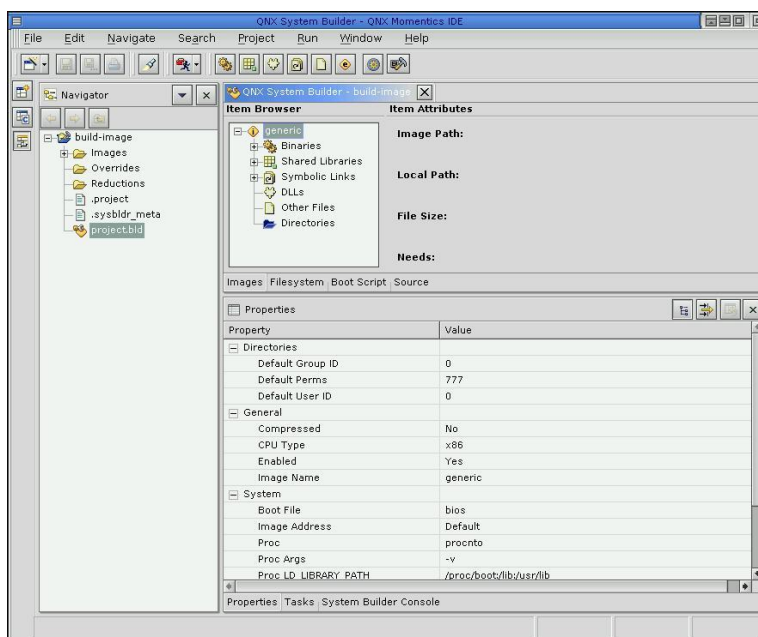


Рис. 1. QNX System Builder

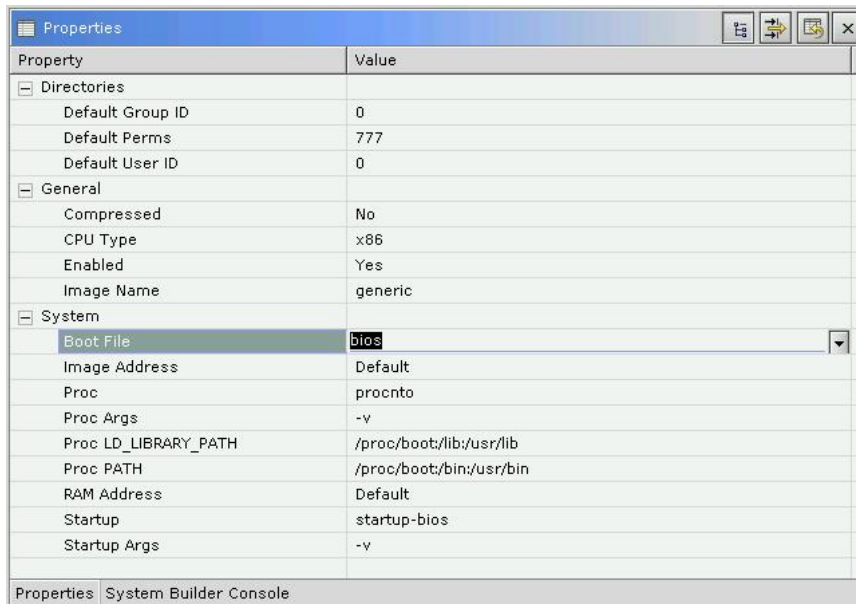


Рис. 2. Редактор властивостей


Тепер додамо в образ необхідні виконавчі файли. Для цього клацнемо на імені образа і або натиснемо відповідну кнопку на панелі інструментів (  ), або на імені образа клацнемо правою кнопкою миші і скористаємося випадним меню. На екрані з'явиться майстер додавання виконавчих файлів (рис. 3).



Рис. 3. Майстер додавання виконавчих файлів

У списку доступних виконавчих файлів, що з'явився, виберемо файл драйвера консолі і натиснемо кнопку Finish. Потім у такий же спосіб додамо файли ls, sin, pidin, ksh. Майстер дозволяє вибирати відразу кілька елементів списку. Усе, що ми додали, з'явиться у вікні **Item Browser** (рис. 4).

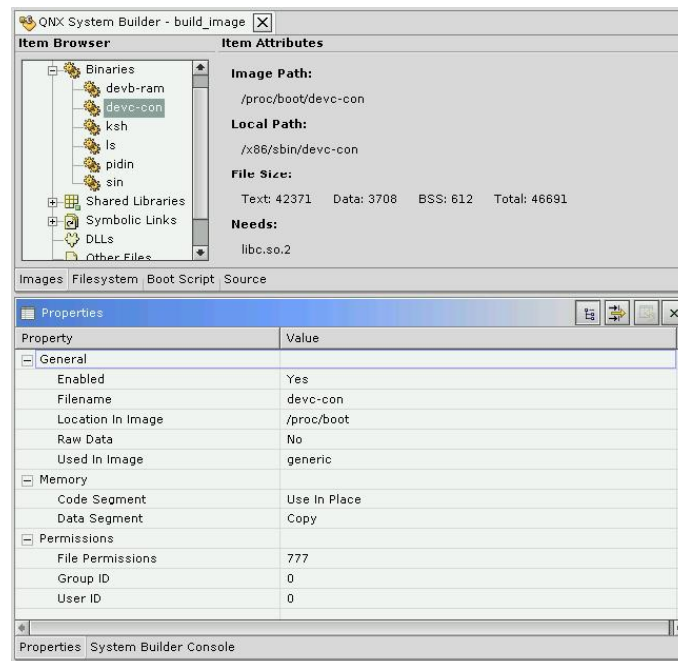



Рис. 4. Item Browser

Треба відмітити, що QNX System Builder автоматично додасть в образ розділювану бібліотеку lib.so. і встановить для неї символічне посилання /usr/lib/ldqnx.so. Ці елементи обов'язково повинні бути присутніми в образі – без них програми, додані в образ, не зможуть виконуватися. Перш ніж генерувати образ, можна скористатися чудовим інструментом – майстром оптимізації "System Optimizer", для чого натиснемо кнопку . Він може виконувати три види оптимізацій: видалити невикористовувані бібліотеки, включені нами в образ, додати необхідні бібліотеки, що ми забули включити в образ і, нарешті, для скорочення розмірів образу майстер може видалити з бібліотек ті функції, що не використовуються (рис. 5).

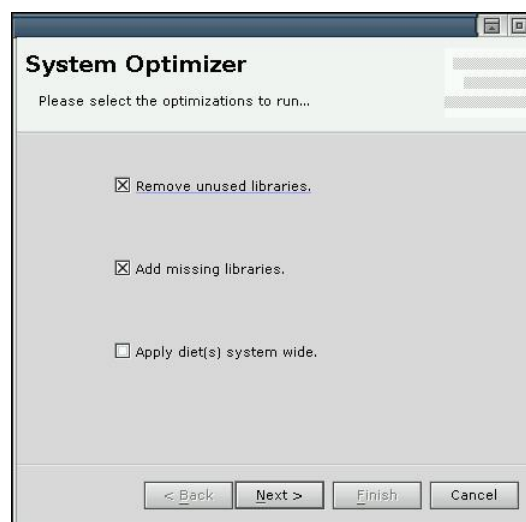


Рис. 5. System optimizer

У нашому випадку образ занадто простий, щоб оптимізація дала які-небудь результати (рис. 6).



Рис. 6. Результат оптимізації

Item Browser дозволяє подивитися, яке дерево каталогів буде створено при завантаженні образу. Інформація про кожен елемент образу відображається у вікні властивостей **Properties** і у вікні атрибутів **Item Attributes** (рис. 7).

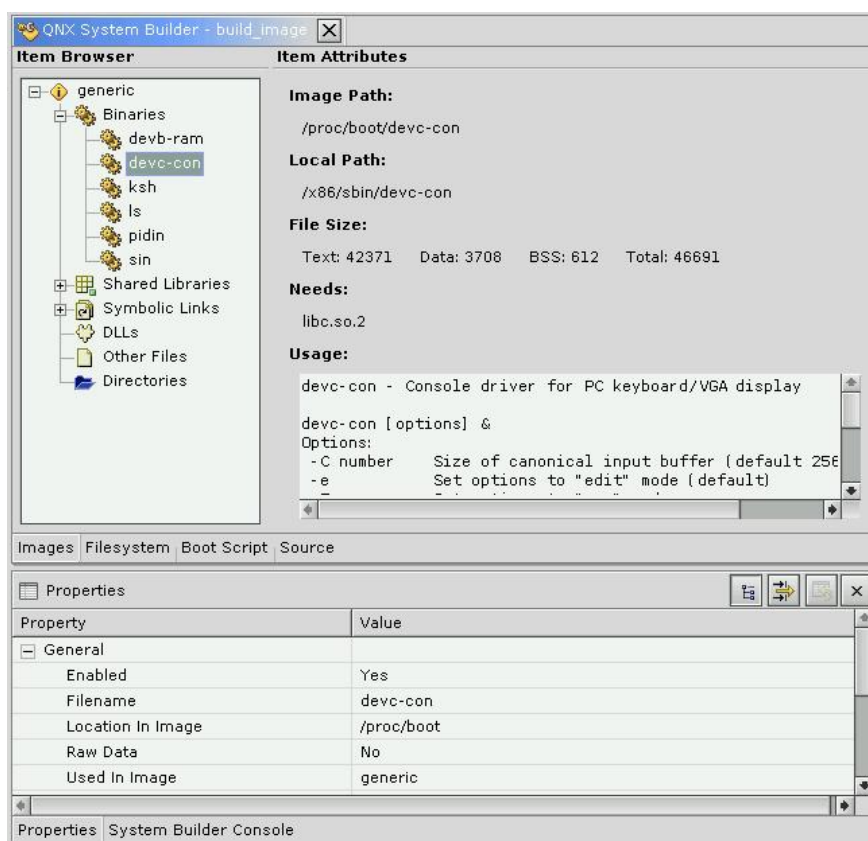


Рис. 7. Item Attributes

Але як вказати модулеві procnto, що робити з програмами, включеними в образ? Для цієї мети існує сценарій завантаження. Відредагуємо цей сценарій, відкривши вкладку **Boot Script** переглядача компонентів, наступним чином(рис. 8).

```
display_msg"" //виводить на екран пусту стрічку повідомлення
```

```

display_msg "Example Image 1" // виводить на екран повідомлення
                               //Example Image 1
PATH=/proc/boot // задається значення змінної оточення PATH
LD_LIBRARY_PATH=/proc/boot:lib/dll // задається значення
змінної                               //оточення LD_LIBRARY_PATH
devc-con -n1 // задається значення кількості відкритих консолей
reopen /dev/con1 // відкриває консоль
[+session] ksh // відкриває на консолі сесію командного
                // інтерпритатора ksh

```

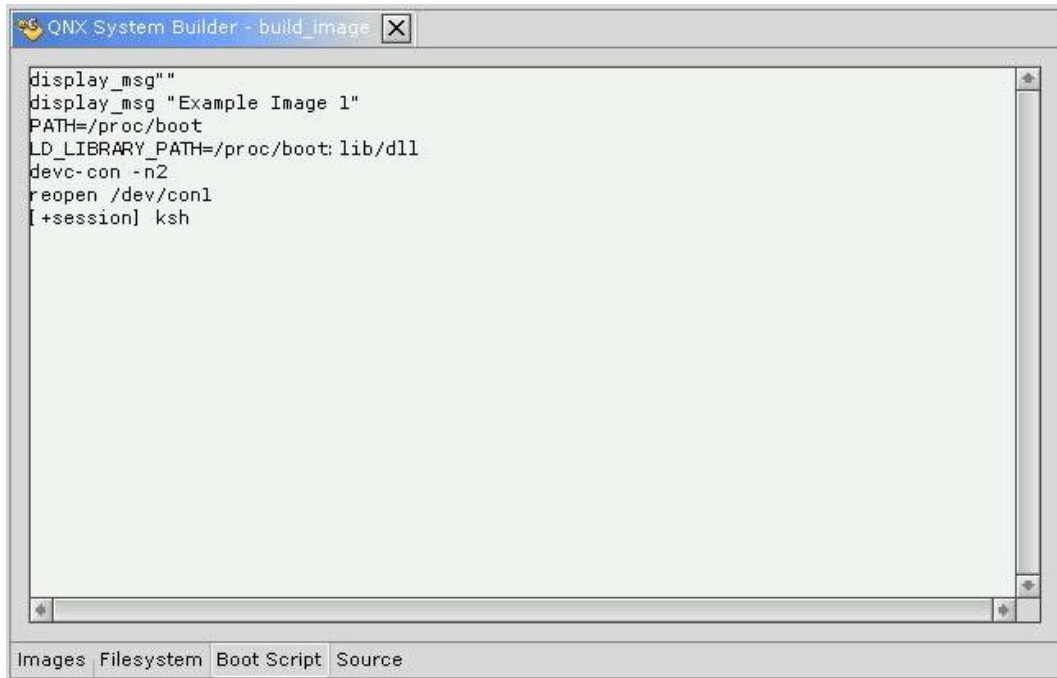


Рис. 8. Boot script

Тепер можна натискати кнопку **Build System** (🔧) на панелі інструментів. У вікні **System Builder Console** з'являться повідомлення утиліти mkifs (рис. 9).



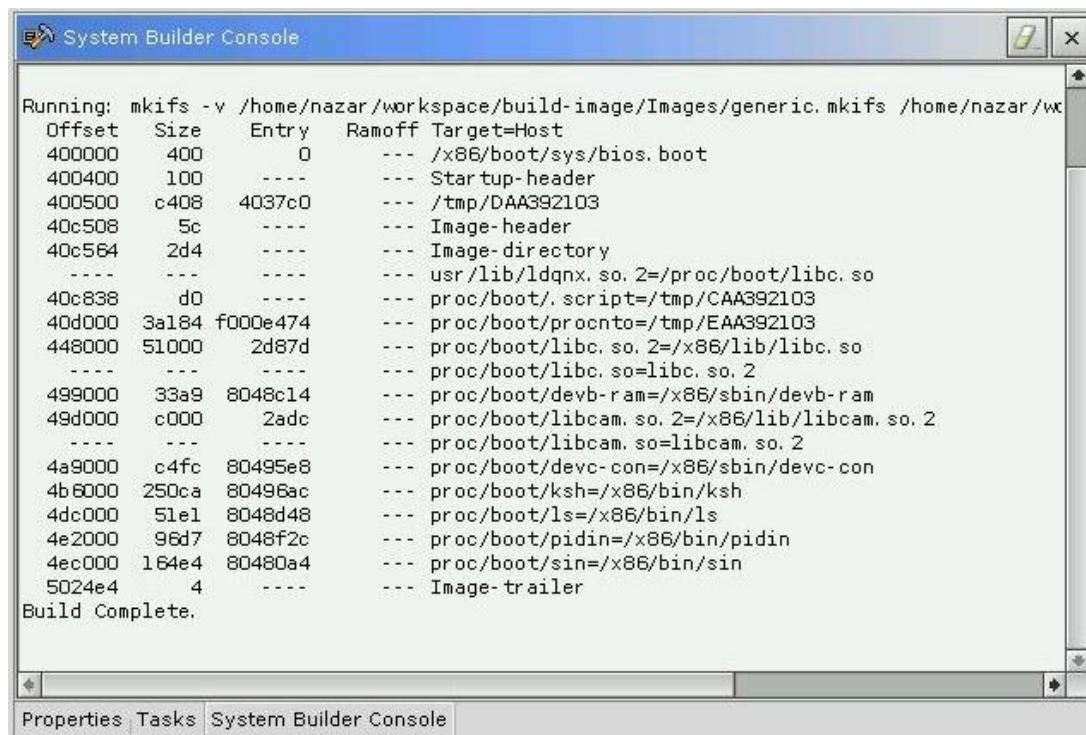


Рис. 9. System Builder Console

Можна, до речі, подивитися build-файл, згенерований IDE, і його властивості – розмір, дату модифікації (рис. 10).

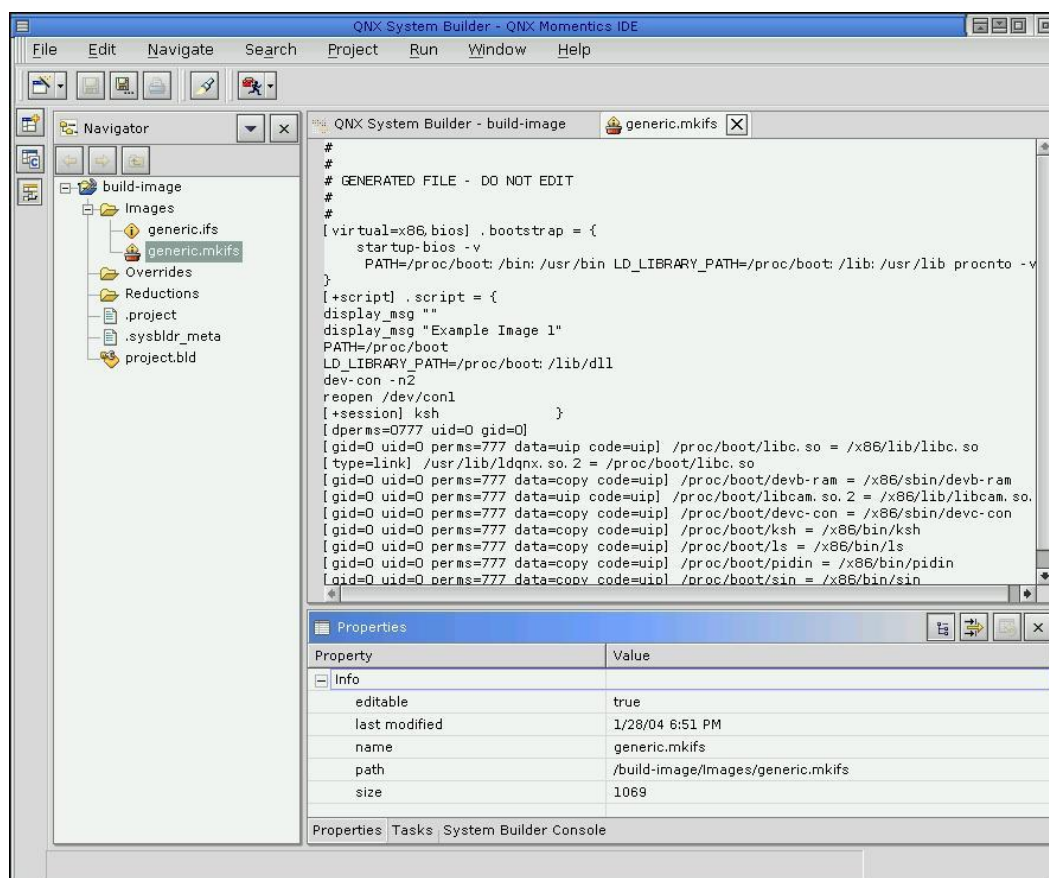


Рис. 10. Властивості згенерованого файлу

Отриманий образ, а він має шляхове ім'я \$HOME/workspace/build\_image/Images/generic.ifs, можна тепер записати, на дискету, відформатовану QNX-утилітою dinit, у файл із іменем boot. Після цього ви зможете завантажитися з цієї дискети і скористатися утилітами ls, pidin, sin, а також вбудованими директивами Korn Shell.

```
#dinit -f /root/workspace/build_image/Images/generic.ifs /dev/fd0
```

Завдання:

1. Запустити систему та зареєструватись.
2. Перейти в графічний режим роботи.
3. Завантажити IDE QNX
4. Виконати побудову власного завантажувального образу.
5. Записати образ на гнучкий магнітний диск.
6. Завершити роботу системи.
7. Здійснити завантаження системи з даного диску.
8. Зупинити роботу системи.



## ЛАБОРАТОРНА РОБОТА № 4

### Тема: Ознайомлення з графічним середовищем розробки Photon Application Builder та набуття навичок для створення найпростіших програм

**Мета:** Ознайомитися з графічним середовищем розробки Photon Application Builder та створити програму для розв'язку арифметичного виразу.

Короткі теоретичні відомості. Середовище PhAB.

Графічний інтерфейс користувача Photon microGUI містить в собі потужний засіб розробки який називається Photon Application Builder (аббревіатура PhAB). За допомогою засобів графічного середовища, розроблювач може проектувати інтерфейсну частину додатка, а також писати програмний код і зв'язувати його з керуючими елементами. Це дає змогу розробнику значно скоротити затрачений час на побудову та відладку програми. Для написання прикладних програм чи додатків PhAB використовує мову програмування C або C++.

Графічне середовище розробки PhAB являє собою багатовіконну систему, яка зображена на рис. 1.

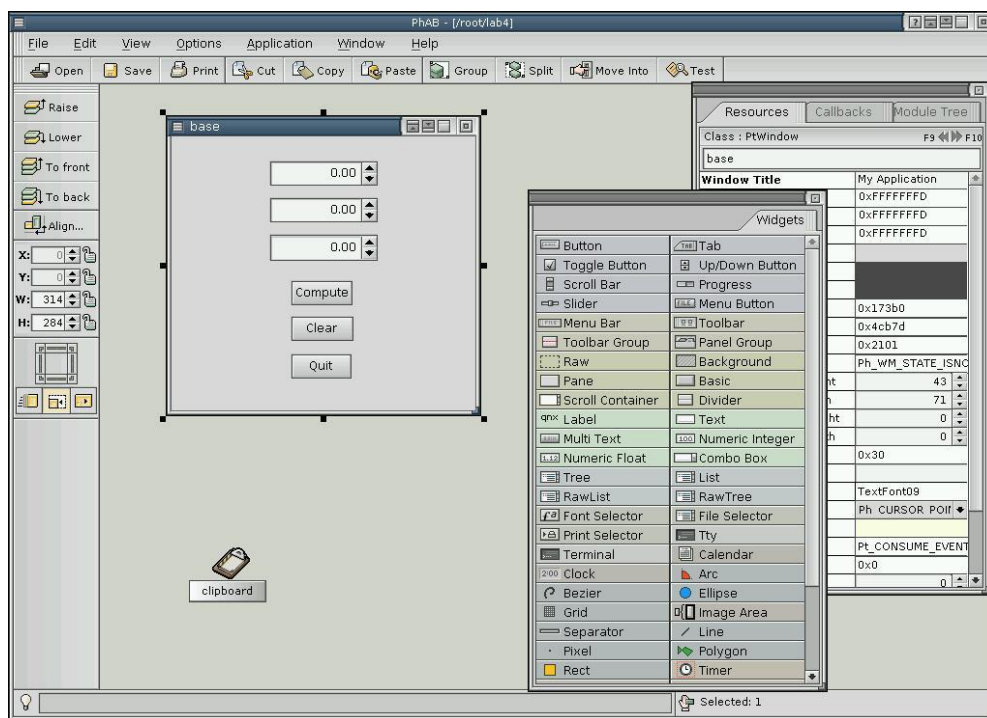


Рис. 1. Графічне середовище розробки PhAB

Панель меню містить великий набір команд для доступу до функцій PhAB  
рис. 2:

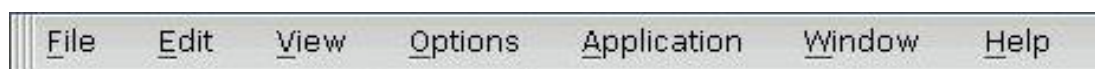


Рис. 2. Панель меню

Команди меню **File** дозволяють створювати вікна, діалоги, генерувати код для розробки прикладних програм, імпортувати графіку, відкривати раніше створені проекти, та зберігати вже існуючі.

Команди меню **Edit** дозволяють виконувати операції з буфером обміну, а також дають можливість вирівнювати групи розміщених на формі компонентів по розмірах і розміщенню, а також створювати та редагувати шаблони.

Команди меню **View** дозволяють викликати на екран різні вікна, необхідні для проектування.

Команди меню **Options** використовуються для задання загальних налаштувань PhAB (відображення сітки, генерування повідомлення по програмі, настройка прав).

Команди меню **Application** дозволяють додавати і забирали з проекту вікна, діалоги, меню, значки і картинки, створювати внутрішні зв'язки між компонентами, задавати опції проектів, генерувати, компілювати і виконувати їх.

Команди меню **Window** використовуються для роботи з вікнами.

Команди меню **Help** дозволяють отримати різноманітну інформацію про графічне середовище Photon та середовище розробки PhAB.

Панель інструментів (рис. 3) містить кнопки для виклику часто використовуваних команд головного меню, з їх допомогою ви можете швидко перемістити, вирівняти, змінити розміри групи компонентів, чи їх певну кількість.

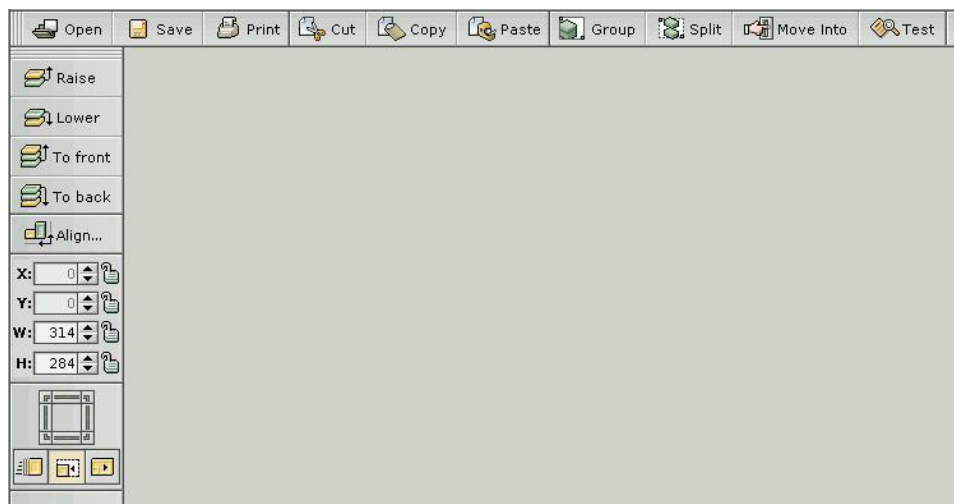


Рис. 3. Панель інструментів

Опишемо призначення кнопок панелі інструментів:

Open (Відкрити) – дозволяє відкрити існуючий додаток;

Save (Зберегти) – дозволяє зберегти поточний додаток;

Print (Друк) – дана команда не впроваджена в системі;

Cut, Copy, Paste (Вирізати, Скопіювати, Вставити) – дозволяють вирізати чи скопіювати компоненти в буфер обміну, і після цього вставити їх з нього.

Group, Split (Згрупувати, Розбити) – дозволяють зкомпонувати виділені компоненти в групу, чи розбити виділену групу на окремі елементи.

Move Into (Перемістити в) – дозволяє перемістити компонент з одної форми на іншу.

Test (Тест) – дозволяє переключитися в режим тестування, який дозволяє взаємодіяти з компонентами так, як при виконанні програми.

Raise, Lower, To Front, To Back (Наперед, назад, на рівень вище, на рівень нижче) – дозволяють перемістити компоненти наперед або назад відносно інших компонентів.

Align (Вирівняти) – містить найбільш часто використовувані команди для вирівнювання виділених компонентів;

X, Y, W, H – вказуються значення координат і розмірів виділених компонентів. Щоб їх змінити потрібно набрати нові значення у відповідних полях і натиснути Enter, щоб заборонити зміну координат і розмірів потрібного компоненту, необхідно клацнути на значку замка, щоб він закрився.

Nudge Tool (Інструменти зсуву) – дані інструменти дозволяють переміщати, збільшувати, стискати компоненти. Для цього потрібно клацнути на відповідній кнопці для вибору потрібного режиму, а далі на відповідній копці для вибору напрямку переміщення чи зсуву (рис. 4).

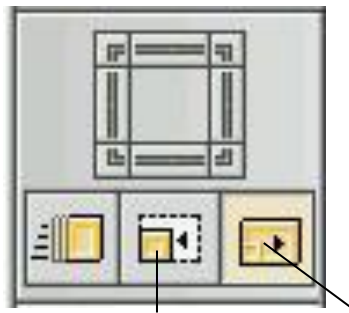


Рис. 4 Інструмент зсуву

Панель керування – PhAB містить набір панелей керування, які відображають інформацію про поточний виділений компонент чи компоненти. Вони відображаються в PhAB по замовчуванню, проте їх можна розмістити будь-де за бажанням користувача. Якщо потрібна панель керування в даний момент закрыта, її можна відкрити вибравши в підменю View (Вигляд) відповідний пункт з її назвою.

Панелі керування містять:

- Палітру компонентів (Widget Palette)
- Панель ресурсів (Resource panel)
- Панель викликів (Callbacks panel)
- Панель дерева модуля (Module Tree panel)

#### **Палітра компонентів.**

Палітра компонентів дозволяє додавати компоненти до додатку користувача.

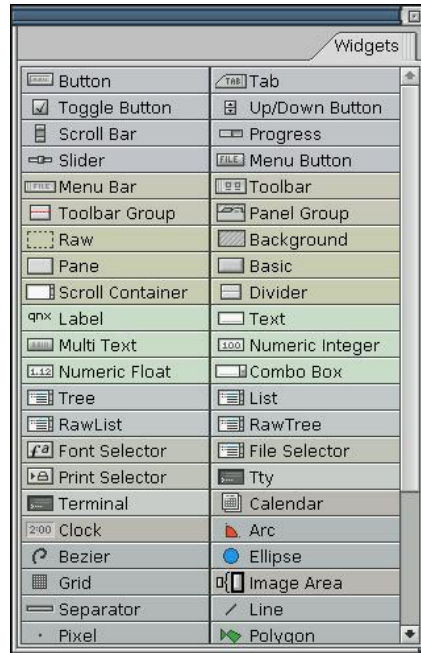


Рис. 5. Палітра компонентів

В даній панелі компоненти впорядковані і розділені різним кольором в групи відносно їх типу. Щоб в даній панелі, компоненти відображалися лише як значки, потрібно клацнути правою клавішею миші на палітрі і вибрати відповідний пункт з випадного списку меню. Щоб знати який саме компонент представляє кнопка панелі, потрібно затримати курсор миші над нею, а ж до появи спливаючої підказки.

Палітра компонентів має два режими:

- Режим вибору, який дозволяє відмічати наявні компоненти і модулі в робочій зоні;
- Режим створення, який дозволяє створювати нові компоненти.

Щоб взнати користувачу в якому режимі він знаходиться потрібно:

- Подивитися на палітру компонентів – якщо кнопка значка натиснута, то користувач знаходиться в режимі створення.
- Подивитися на вказівник миші – якщо він має вигляд одинарної стрілки коли перемістити його в робочу зону, тоді користувач знаходиться в режимі вибору, якщо ж він має інший вигляд, тоді в режимі створення.

Щоб перейти з режиму створення в режим вибору необхідно або клацнути вказівником на фоні робочої зони, або клацнути правою клавішею миші на формі, або клацнути на виділеному компоненті в палітрі компонентів.

По замовчуванню, PhAB переходить в режим вибору після створення компоненту.

### **Панель ресурсів.**

Панель ресурсів відображає список ресурсів (властивостей) виділеного компоненту чи компонентів. (Якщо виділені більше ніж один компонент то будуть відображатися спільні для них властивості).



Рис. 6. Панель ресурсів

Вона містить наступне:

Клас компоненту – відображає до якого класу відноситься виділений компонент;

Кнопки наступних і попередніх компонентів – дозволяють послідовно переходити від одного до другого компонента модуля;

Назва екземпляру – дозволяє задати унікальну назву екземпляра компонента.

Користувач може змінювати значення властивості в правій частині панелі керування або використовувати повноцінний редактор клацнувши на назві властивості.

### Панель викликів.

Панель викликів відображає список ресурсів викликів для виділеного компоненту. Користувач може використовувати цю панель лише тоді, коли виділений лише один компонент. Компонент також має мати унікальну назву екземпляру.



Рис. 7. Панель викликів

Ця панель як і панель ресурсів відображає клас компоненту, назву екземпляру і кнопки переходу між компонентами.

Ліва сторона списку вказує тип виклику. Права сторона відображає:

- “None” немає викликів;
- тип виклику і його ім’я, якщо він один;
- кількість викликів, якщо їх більше одного.

Щоб створити виклик чи відредагувати вже існуючий, потрібно клацнути на відповідному ресурсі (наприклад, Pt\_CB\_ACTIVATE).

### Панель дерева модуля.

Панель дерева модуля відображає ієрархічне дерево компонентів даного модуля.

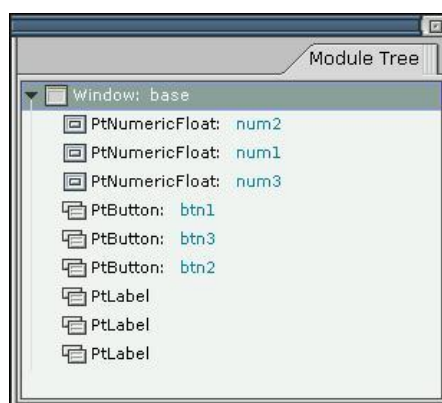


Рис. 8. Панель дерева модуль

Ця панель робить легким:

- перегляд компонентів модуля в відношенні батько/нащадок;
- виділення компонентів всередині групи;
- пошук компоненту по назві;
- виділення компоненту прихованого під іншим компонентом.



Щоб виділити компонент з дерева, необхідно клацнути лівою клавішею миші на його назві. Щоб викликати для нього меню, необхідно клацнути правою клавішею миші на ньому.

Приклад створення програми.

Опишемо покроково створення програми для обчислення заданої функції –  $y=x1+x2$ , та виводу результату.

Для цього необхідно виконати наступні дії:

1. Запускаємо PhAB.
2. З меню File вибираємо пункт New, відкриється вікно (малюнок) в якому необхідно буде вибрати потрібну форму, виберемо напростішу Plain і натискаємо клавішу Done.

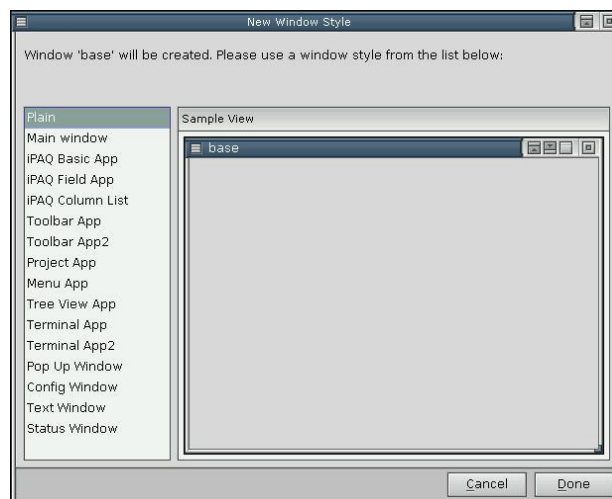


Рис. 9. Панель створення форми

3. В панелі ресурсів для компонента PtWindow: base міняємо властивість Window Title – на lab4.
4. Вибираючи необхідні компоненти із палітри компонентів, будуємо наступну форму для нашої програми

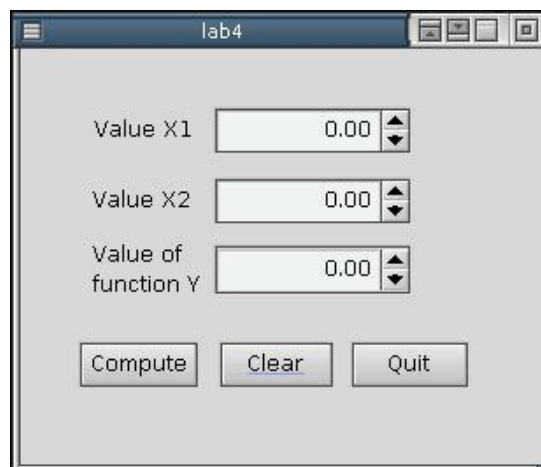


Рис. 10. Компоненти робочої форми

4. Для текстових елементів Value X1, Value X2 та Value of function Y вибираємо компонент Label, та ставимо їх на форму. В панелі ресурсів змінюємо для цих

компонентів властивість *Text* – на *Value X1*, *Value X2* та *Value of function Y* відповідно.

5. Для формування вікон вводу значень змінних *X1* та *X2* вибираємо компонент *Numeric Float*, та ставимо їх на форму. В *Resource Panel* змінюємо властивість *Instance Name* – на *num1* та *num2* відповідно.
6. Для відображення результату обчислення функції вибираємо також компонент *Numeric Float* і ставимо його на форму. В *Resource Panel* змінюємо властивість *Instance Name* на *num3*.
7. Для формування клавiш управління роботою програми (*Compute*, *Clear*, *Quit*) – вибираємо компонент *Button* та ставимо три його екземпляри на форму. В *Resource Panel* змінюємо властивість *Instance Name* на *btn1*, *btn2* і *btn3*, відповідно до клавiш. Також відповідно до клавiші, змінюємо властивість *Text* на *Compute*, *Clear*, *Quit*.

Щоб програма працювала необхідно задати та описати дії, які вона буде виконувати, коли користувач натискатиме відповідні клавiші та вводити данні. Для цього запрограмуємо кожну клавiшу на виконання властивої їй функції.

8. Для задання функції, яка буде виконуватися при натиску на клавiші *Compute*, в *Callback Panel* для цієї клавiші, клацнемо в полі *Activate*, з'явиться вікно зображене на рисунку 11, в якому, в полі *Function* потрібно ввести назву функції (назвемо її *compute*). Після цього потрібно клацнути в полі *Code Types* на клавiші *Code*, це буде означати, що дана функція буде містити лише програмний код. І на кінець потрібно натиснути клавiші *Apply* і *Done*, щоб зберегти зміни і закрити вікно.

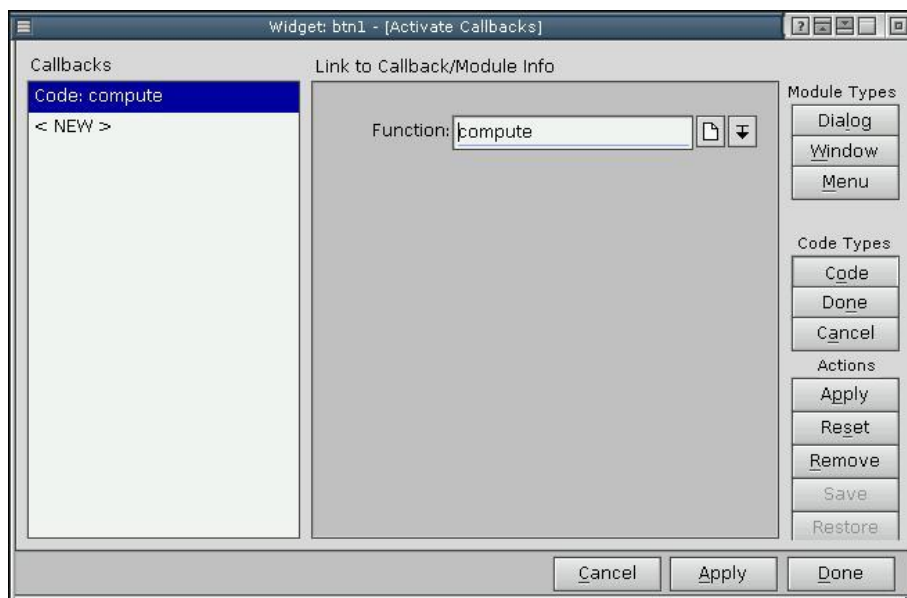


Рис. 11. Створення виконавчої функції

9. У відповідності до попереднього пункту, задамо виконавчі функції (*clear* і *quit*) для клавiш *Clear* і *Quit*.
10. Далі потрібно записати нашу програму, для цього вибираєм на панелі інструментів чи з головного меню команду *Save As...*, з'явиться вікно, що зображене на рисунку 12, у полі *Name* потрібно вказати ім'я папки в якій буде зберігатися програма.



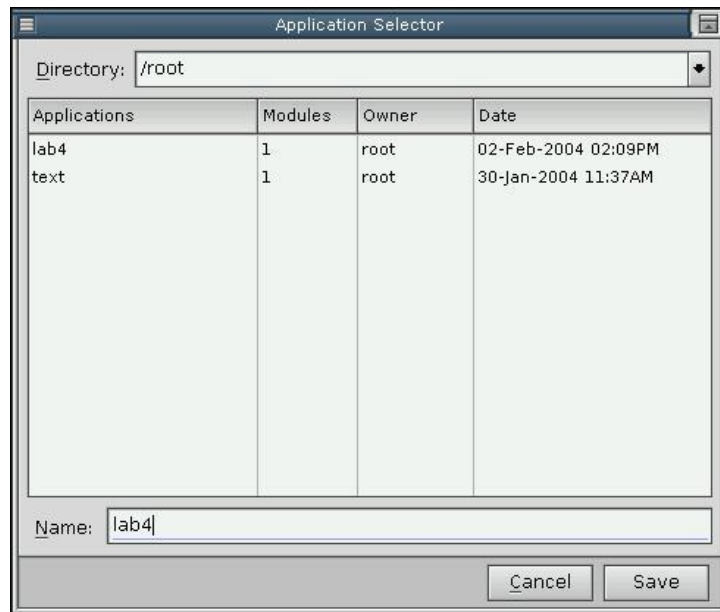


Рис. 12. Вікно збереження робочої програми

11. Після цього потрібно скомпонувати програму для відповідної процесорної платформи. Для цього в підменю Application вибираємо пункт Build + Run, з'явиться вікно зображене на рисунку 13. У ньому клацаємо на клавіші Generate. У наступному вікні що з'явиться, вибираємо ntox86.gcc і тиснемо клавішу Generate. Після закінчення процесу генерування додатку тиснем на клавіші Done.

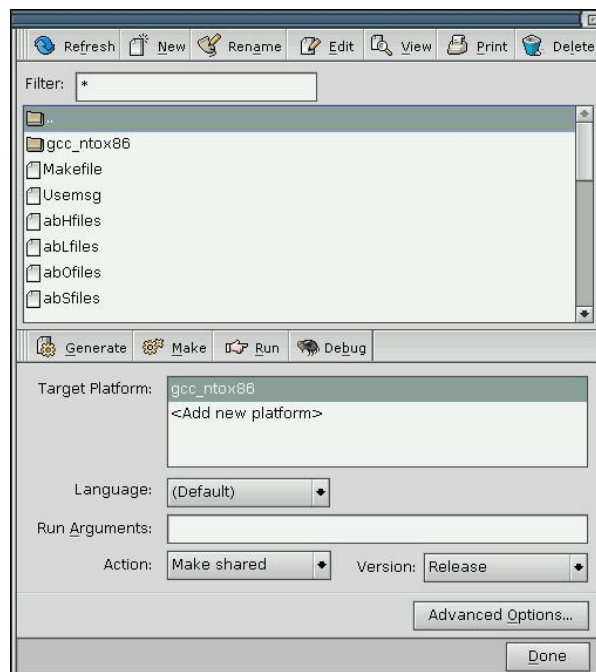


Рис. 13. Копановка програми

12. Далі в списку файлів вибираємо створену нами функцію compute.c (розширення вказує на те що функція написана на мові C), яка керує роботою клавіші compute, і двічі клацаєм на її назві чи вибираємо команду Edit з меню

вікна, відкриється вікно редактора коду. Змінимо текст функції так як показано нижче.

```
/* Your Description */
/* AppBuilder Photon Code Lib */
/* Version 2.01 */

/* Standard headers */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>

/* Local headers */
#include "ablibs.h"
#include "abimport.h"
#include "proto.h"

int
compute( PtWidget_t *widget, ApInfo_t *apinfo, PtCallbackInfo_t *cbinfo )

{
    double *x1, *x2, y;
    /* eliminate 'unreferenced' warnings */
    widget = widget, apinfo = apinfo, cbinfo = cbinfo;
    PtGetResource( ABW_num1, Pt_ARG_NUMERIC_VALUE, &x1, 0 );
    PtGetResource( ABW_num2, Pt_ARG_NUMERIC_VALUE, &x2, 0 );
    y = *x1 + *x2;
    PtSetResource( ABW_num3, Pt_ARG_NUMERIC_VALUE, &y, 0 );
    return( Pt_CONTINUE );
}
```

13. Відповідно до попереднього пункту редагуємо код функцій clear та quit.

```
/* Your Description */
/* AppBuilder Photon Code Lib */
/* Version 2.01 */

/* Standard headers */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>

/* Local headers */
#include "ablibs.h"
#include "abimport.h"
```

```
#include "proto.h"
```

```
int
```

```
clear( PtWidget_t *widget, ApInfo_t *apinfo, PtCallbackInfo_t *cbinfo )
```

```
{
```

```
/* eliminate 'unreferenced' warnings */
```

```
widget = widget, apinfo = apinfo, cbinfo = cbinfo;
```

```
PtSetResource( ABW_num1, Pt_ARG_NUMERIC_VALUE, 0, 0 );
```

```
PtSetResource( ABW_num2, Pt_ARG_NUMERIC_VALUE, 0, 0 );
```

```
PtSetResource( ABW_num3, Pt_ARG_NUMERIC_VALUE, 0, 0 );
```

```
return( Pt_CONTINUE );
```

```
}
```

```
/* Your Description */
```

```
/* AppBuilder Photon Code Lib */
```

```
/* Version 2.01 */
```

```
/* Standard headers */
```

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
#include <unistd.h>
```

```
#include <string.h>
```

```
/* Local headers */
```

```
#include "ablibs.h"
```

```
#include "abimport.h"
```

```
#include "proto.h"
```

```
int
```

```
quit( PtWidget_t *widget, ApInfo_t *apinfo, PtCallbackInfo_t *cbinfo )
```

```
{
```

```
/* eliminate 'unreferenced' warnings */
```

```
widget = widget, apinfo = apinfo, cbinfo = cbinfo;
```

```
PtExit( EXIT_SUCCESS );
```

```
return( Pt_CONTINUE );
```

```
}
```

14. Після цього потрібно скомпілювати нашу програму, для цього у попередньому вікні вибираємо команду Make.

15. Якщо процес компіляції пройшов без помилок, так як показано на рисунку 14, програму можна запустити на виконання командою Run.



Рис. 14. Процес компіляції

Таким чином, було створено програму для обчислення заданого математичного виразу за допомогою середовища PhAB.

**Завдання.** Скласти програму для обчислення заданого арифметичного виразу згідно варіанту. Програма повинна забезпечити ввід необхідних даних та вивід результату.

Варіант	Завдання
1	$y = \sqrt{\frac{x_2^2 + x_1 / x_2}{16x_2x_1}}$
2	$y = 23 \cos^2(x_1^3 x_2^5) + 2x_1$
3	$y = \sin^2\left(x_1 \frac{x_2}{x_1 + 53x_2^2}\right)$
4	$y = 45x_1 \sin x_2 + \sqrt{9x_2x_1^3}$
5	$y = 0.1x_1 \sin x_2 \cos x_1^4 + 55$
6	$y = \operatorname{tg}(x_1 - x_2^2) + 31.55x_2x_1^2$
7	$y = \sin(x_1 - x_2^3 + \sqrt{x_1}) - 1.3x_1^3$
8	$y = \cos(\sqrt{x_2} + 34x_1) - 4\sin(x_2)$
9	$y = \frac{4\sin(3 + x_1x_2)}{34 - 9x_2^3}$
10	$y = \frac{\cos^2\left(\lg_{10} \frac{x_2}{x_1}\right)}{45 + x_2}$
11	$y = \cos^3\left(\lg_{10} \frac{x_1 + 2x_2 + 9}{0.666}\right)$
12	$y = \sqrt{56 + \frac{x_1 + x_2 + \sin(x_1x_2)}{5\cos(x_2^2)}}$
13	$y = 45 \sin(x_1 + x_2 + \lg_{10}(x_1x_2^2))$
14	$y = \frac{\sin^3(x_1) + 45 + x_2}{2x_1^4 + 4x_2}$
15	$y = \frac{\sqrt{x_1^3 + x_2^5}}{1000} + 65$

Щоб скористатися готовими математичними функціями, потрібно підключити бібліотеку `math.h`. Для цього потрібно на початку файлу в розділі опису стандартних заголовків функції `compute`, додати стрічку:

```
#include <math.h>.
```

У програмі можуть бути використані такі математичні функції:

`double cos( double )` – обчислює косинус кута;

`double sin( double )` – обчислює синус кута;

`double tan( double )` – обчислює тангенс кута;

`double log10( double )` – обчислює логарифм десятковий числа;

`double sqrt( double )` – обчислює корінь квадратний числа;

`double pow( double x, double y )` – здійснює піднесення до степеня у, числа х.

## Лабораторна робота № 5

### Тема: Проектування систем реального часу на базі операційної системи QNX-6

**Мета:** Вивчення особливостей проектування та освоєння спеціальних засобів розробки систем реального часу та їх компонентів на базі ОС QNX-6.

Складання та редагування відповідних текстів програм, їх компіляція та завантаження на виконання. Розробка однопотокових та багатопотокових консольних і графічних додатків. Розробка фрагментів систем реального часу за заданими часовими характеристиками.

#### 1. Загальна характеристика ОС QNX-6

Операційна система QNX-6, яку також називають QNX/Neutrino2 (Neutrino2 – назва мікроядра) або QNX RealTime Platform [3], призначена для розробки проектів систем реального часу на різних апаратних платформах. Операційна система QNX-6 має клієнт-серверну архітектуру та складається з мікроядра і згрупованих довкола нього взаємодіючих процесів [3]. Безпосередньо мікроядро надає тільки декілька сервісів, а саме: планує потоки, подає сигнали, передає повідомлення, виконує синхронізацію потоків та їх диспетчеризацію та обслуговує таймер. Тому мікроядро має невеликий розмір, причому мікроядро саме по собі не піддається диспетчеризації. Його код виконується тільки в результаті виклику певної системної функції чи в результаті виникнення апаратного переривання. Додаткові функціональні можливості ОС QNX-6 забезпечуються серверними процесами (серверами), які мають відповідати на запити клієнтських процесів і які не входять до складу мікроядра. Типова конфігурація ОС QNX-6 має наступні стандартні серверні процеси: менеджер процесів (procnto), менеджер файлової системи (fs), менеджер пристроїв (dev), менеджер мережі (qnet). Зазначені менеджери в рамках термінології ОС QNX-6 часто називають адміністраторами ресурсів.

В QNX-6 мікроядро діє як програмна шина, яка дозволяє динамічно підключати різні менеджери кожен раз, коли вони потрібні. Загальна структура операційної системи QNX-6 зображена на рис.1. Слід зазначити, що зазвичай менеджер процесів procnto та мікроядро об'єднуються в один програмний модуль, що керує всіма процесами в ОС QNX-6.

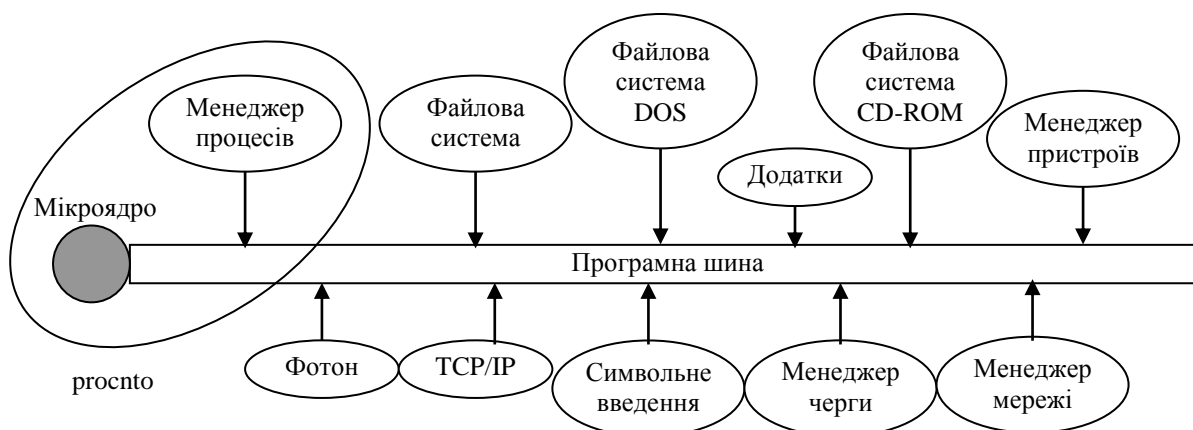


Рис. 1. Архітектура ОС QNX-6

Мікроядро запускається на 0 (нульовому) рівні привілеїв процесора, а менеджери і драйвери пристроїв запускаються на рівнях 1 і 2 (для виконання дій введення/виведення). Прикладні процеси запускаються на 3 (третьому) рівні привілеїв і тому можуть виконувати тільки загальні команди процесора.

QNX-6 – це операційна система, яка заснована на механізмі передачі повідомлень. Цей механізм є фундаментальним засобом реалізації взаємодії процесів поміж собою, тобто він забезпечує так званий IPC - InterProcess Communication. Механізм передачі повідомлень заснований на моделі “клієнт-сервер”: клієнт надсилає повідомлення серверу, який потім повертає результат. Зрозуміло, що механізм передачі повідомлень повинен бути (і є в QNX-6) прозорим щодо мережі, і тому розроблювана система може бути розподілена між декількома мережевими вузлами без будь-яких змін у коді розробленого програмного забезпечення.

Для передачі повідомлень між клієнтом і сервером, QNX-6 використовує механізм пріоритетів, керований клієнтом. Це означає, що серверний процес успадковує рівень пріоритету клієнтського процесу, який вимагає обслуговування. Коли обслуговування запиту клієнта завершено, серверний процес може відновити свій первісний рівень пріоритету. Якщо обслуговування вимагають декілька клієнтів водночас, серверний процес приймає рівень пріоритету клієнтського процесу з найбільш високим пріоритетом.

Архітектура клієнт-сервер має багато переваг, одна з яких – стійкість до помилок. Так, кожен серверний та клієнтський процеси завантажуються в своїх власних адресних просторах віртуальної пам'яті, що приводить до надійності і стійкості розробленої системи в цілому, але виконання системних викликів потребує переключення контексту активної задачі, що приводить до додаткових витрат на захист пам'яті та до зниження продуктивності роботи. Для підвищення зазначеної продуктивності роботи розробку програмного коду процесів (як серверних, так і клієнтських) зазвичай виконують в стилі багатопотокових додатків.

Додаткову інформацію про загальні характеристики операційної системи QNX-6 та їх порівняння з характеристиками інших операційних систем, придатних до розробок систем реального часу, можна знайти в [4-7].

## ***2. Принципи роботи мікроядра QNX-6.***

Мікроядро операційної системи QNX-6, яке має назву Neutrino2, реалізує основні сервіси операційної системи та багатопотоковість безпосередньо у мікроядрі, і тому такі сервіси доступні без наявності додаткових модулів. Мікроядро Neutrino2 підтримує такі сервіси (об'єкти):

- потоки,
- повідомлення,
- сигнали,
- годинник,
- таймери,
- програми обробки переривань,
- семафори,
- взаємно-виключні блокування (mutexes),
- умовні змінні (condvars),

- бар'єри.

Сервіси, які включені до складу мікроядра, обрані за коротким шляхом виконання. Інші сервіси, які потребують істотну роботу (наприклад завантаження процесу на виконання), реалізовані зовні мікроядра на різних процесах чи потоках. Так, менеджер процесів `procnto` розширює сервіс, наданий ядром `Neutrino2`, забезпечуючи підтримку процесів, захист пам'яті та адміністрування області файлових шляхів, додаючи при цьому тільки 32 кілобайти коду.

## 2.1. Потоки і процеси в QNX-6

При створенні системи реального часу часто є необхідним паралельне виконання декількох процедур водночас. В мікроядрі `Neutrino2` паралелізм роботи досягається завдяки використанню моделі потоків за стандартом POSIX (Portable Operating System Interface for Unix), який визначає процес як контейнер, що містить в собі один чи більшу кількість потоків виконання.

Потік – це найменша одиниця, що підлягає виконанню і диспетчеризації.

Процес – це "контейнер" для потоків, який визначає адресний простір, у межах якого потоки виконуються. Процес завжди містить в собі принаймні один потік.

В залежності від характеру розроблюваного додатку, потоки можуть виконуватися незалежно один від одного та без потреби зв'язуватися між собою, або вони можуть бути сильно зв'язані. Щоб забезпечити зв'язок і синхронізацію взаємодіючих потоків, `Neutrino2` надає різноманітні сервіси IPC і синхронізації.

Для роботи з потоками існує бібліотека `pthread` (потоки POSIX), яка не входить до складу мікроядра і яка має наступні функції:

```
pthread_attr_destroy ()
pthread_attr_getdetachstate ()
pthread_attr_getinheritsched ()
pthread_attr_getschedparam ()
pthread_attr_getschedpolicy ()
pthread_attr_getscope ()
pthread_attr_getstackaddr ()
pthread_attr_getstacksize ()
pthread_attr_init ()
pthread_attr_setdetachstate ()
pthread_attr_setinheritsched ()
pthread_attr_setschedparam ()
pthread_attr_setschedpolicy ()
pthread_attr_setscope ()
pthread_attr_setstackaddr ()
pthread_attr_setstacksize ()
pthread_cleanup_pop ()
pthread_cleanup_push ()
pthread_equal ()
pthread_getspecific ()
pthread_setspecific ()
pthread_testcancel ()
pthread_key_create ()
```



pthread\_key\_delete ()  
pthread\_once ()  
pthread\_self ()  
pthread\_setcancelstate ()  
pthread\_setcanceltype ()

Таблиця 1 подає POSIX-функції потоків, що входять до складу мікроядра.

Таблиця 1

POSIX запит	Запит мікроядра	Опис
pthread_create ()	ThreadCreate ()	Створити новий потік для виконання
pthread_exit ()	ThreadDestroy ()	Знищити потік
pthread_detach ()	ThreadDetach ()	Відокремити потік
pthread_join ()	ThreadJoin ()	Приєднати до потоку, очікуючи його завершення
pthread_cancel ()	ThreadCancel ()	Скасувати потік у точці скасування
-	ThreadCtl ()	Змінити Neutrino-визначені характеристики потоку
pthread_mutex_init ()	SyncTypeCreate ()	Створити mutex
pthread_mutex_destroy ()	SyncDestroy ()	Знищити mutex
pthread_mutex_lock ()	SyncMutexLock ()	Блокувати mutex
pthread_mutex_trylock ()	SyncMutexLock ()	Умовно блокувати mutex
pthread_mutex_unlock ()	SyncMutexUnlock ()	Розблокувати mutex
pthread_cond_init ()	SyncTypeCreate ()	Створити змінну умови
pthread_cond_destroy ()	SyncDestroy ()	Знищити змінну умови
pthread_cond_wait ()	SyncCondvarWait ()	Очікувати на змінну умови
pthread_cond_signal ()	SyncCondvarSignal ()	Повідомити про змінну умови
pthread_cond_broadcast ()	SyncCondvarSignal ()	Передати за сигналом змінну умови
pthread_getschedparam ()	SchedGet ()	Одержати параметри диспетчеризації потоку
pthread_setschedparam ()	SchedSet ()	Встановити параметри диспетчеризації потоку
pthread_sigmask ()	SignalProcMask ()	Одержати або встановити маску сигналів потоку
pthread_kill ()	SignalKill ()	Послати сигнал завершення потоку.

Хоча потоки в межах процесу спільно використовують адресний простір процесу, кожен потік ще має й свої локальні дані. У деяких випадках ці дані захищені в межах ядра, у той час як інші локальні дані постійно знаходяться незахищеними в адресному просторі процесу. Нижче приведені деякі типи локальних ресурсів потоку:

- `tid` – кожен потік ідентифікований цілочисленим значенням, що починається з 1 і є унікальним у межах процесу, де існує потік;
- набір реєстрів – кожен потік має свій власний лічильник програми, вказівник вершини стеку, і слово процесора;
- стек – кожен потік виконується в своєму власному стеку, збереженому в межах адресного простору його процесу;
- маска сигналу – кожен потік має власну маску сигналу.

Потік має визначену системою область даних – «thread local storage» (TLS). TLS використовуються, щоб зберегти інформацію про потік (наприклад `tid`, `pid`, вказівник стеку). TLS безпосередньо не доступний додаткові. Дані, які збережені у TLS, реалізовані в бібліотеці `pthread` і забезпечують механізм для зв'язку глобальної змінної процесу з даними потоку. У таблиці 2 наведені функції, що використовуються для створення і керування цими даними.

Таблица 2

Функція	Опис
<code>pthread_key_create ()</code>	Створити ключ даних з функцією деструктора
<code>pthread_key_delete ()</code>	Знищити ключ даних
<code>pthread_setspecific ()</code>	Зв'язати значення даних із ключем даних
<code>pthread_getspecific ()</code>	Повернути значення даних, зв'язане з ключем даних

## 2.2. Життєвий цикл потоку

Кількість потоків у межах процесу може змінюватись протягом виконання програми. Це пов'язано з можливістю динамічно створювати і знищувати потоки. Створення потоку за допомогою виклику функції `pthread_create()` визначає розподіл і ініціалізацію необхідних для потоку ресурсів у межах адресного простору процесу і початок виконання потоку в певному адресному просторі. Завершення потоку за допомогою виклику функцій `pthread_exit ()`, `pthread_cancel ()` визначає зупинку потоку і звільнення ресурсів потоку.

На протязі часу свого існування потік може знаходитися в 3 основних станах: виконуватися (RUNNING), бути готовим до виконання (READY), бути блокованим. Причин, з яких потік може стати блокованим, є багато, і тому визначена певна кількість станів блокування з відповідними назвами (рис.2). Зазвичай після розблокування потік потрапляє до стану READY.

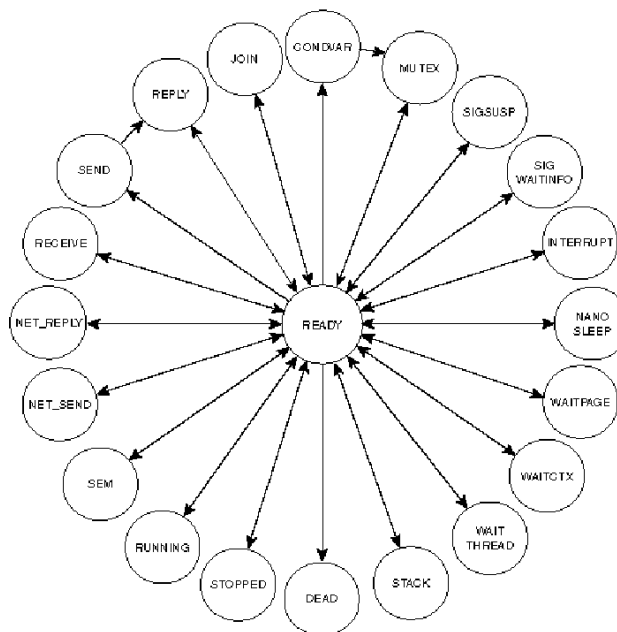


Рис. 2. Стани потоку

Зазначені назви станів означають наступне:

CONDVAR – потік блокований на умовній змінній (при виклику `pthread_condvar_wait ()`).

DEAD – потік закінчив виконання й очікує приєднання другого потоку, не звільняє ресурси.

INTERRUPT – потік блокований, очікує переривання (при виклику `InterruptWait ()`).

JOIN – потік блокований, очікує на приєднання до іншого потоку (при виклику `pthread_join ()`).

MUTEX – потік блокований на взаємно-виключному блокуванні (при виклику `pthread_mutex_lock ()`).

NANOSLEEP – потік не діє протягом короткого інтервалу часу (при виклику `nanosleep ()`).

NET\_REPLY – потік очікує відповідь, що буде отримана через мережу (при виклику `MsgReply * ()`).

NET\_SEND – потік очікує на імпульс або сигнал, що буде отриманий через мережу (при виклику `MsgSendPulse ()`, `MsgDeliverEvent ()`, чи `SignalKill ()`).

READY – потік очікує на виконання, у той час як процесор виконує інший потік такого ж або більш високого пріоритету.

RECEIVE – потік блокований, очікує повідомлення (при виклику `MsgReceive ()`).

REPLY – потік блокований, очікує відповідь на своє повідомлення (при виклику `MsgSend ()` і сервер одержав повідомлення).

RUNNING – потік виконується процесором.

SEM – потік очікує на семафорі (при виклику `SyncSemWait ()`).

SEND – потік блокований на відправленні повідомлення (при виклику `MsgSend ()`, але сервер не одержав повідомлення).

SIGSUSPEND – потік блокований, очікуючи сигнал (при виклику `sigsuspend ()`).

SIGWAITINFO – потік блокований, очікуючи сигнал (при виклику `sigwaitinfo ()`).

STACK – потік очікує віртуальний адресний простір, що буде розподілено для стека потоку (батько викликає ThreadCreate ()).

STOPPED – потік блокований, очікує сигнал продовження SIGCONT.

WAITCTX – потік очікує на неціле число, щоб стати доступним для використання.

WAITPAGE – потік очікує на фізичну пам'ять, що буде розподілена для віртуальної адресації.

WAITTHREAD – потік очікує на закінчення створення дочірнього потоку (при виклику ThreadCreate ()).

### 2.3. Диспетчеризація потоків

Виконання потоку припиняється кожного разу, коли мікроядро одержує системний запит або коли виникає виключення (збій) чи апаратне переривання. Диспетчеризація відбувається кожен раз, коли стан будь-якого потоку змінюється, незалежно від того, який потік у даний момент виконується. Рішення про те, який потік має одержати керування, приймає Neutrino. Якщо керування одержує потік, відмінний від поточного, то виконується переключення контексту активної задачі. Переключення контексту виконується у тому випадку, якщо поточний потік:

- блокується (наприклад, у результаті виклику функції очікування повідомлень)
- добровільно передає керування (наприклад, за допомогою виклику функції sched\_yield())
- якщо в системі існує потік, виконання якого є більш важливим.

Блокований потік видаляється з черги готових потоків і готовий потік з найвищим пріоритетом буде виконуватися. Коли блокований потік розблокується, він буде поміщений у кінець черги готових потоків такого ж рівня пріоритету, як і він сам.

Якщо потік добровільно звільняє процесор (викликає функцію sched\_yield ()), то він ставиться до кінця черги для свого пріоритету. Далі виконується потік з найвищим пріоритетом (це може бути й потік, що тільки що звільнив процесор).

Потік, що виконується, переривається, коли потік з більш високим пріоритетом ставиться в чергу. Перерваний потік залишається на початку черги для свого пріоритету і більш пріоритетний потік виконується.

Існує спеціальний неактивний потік у менеджері процесів, що має пріоритет 0 і завжди готовий працювати.

Потік успадковує пріоритет його батьківського потоку за замовчуванням.

Потік має реальний пріоритет і ефективний пріоритет, і може змінювати їх, але ефективний пріоритет може змінюватися через пріоритетне спадкування або політику диспетчеризації. Звичайно ефективний пріоритет дорівнює реальному пріоритету.

Фактично черга потоків, що готові до виконання, створена як 64 окремі черги, одна для кожного пріоритету. Для забезпечення більшої гнучкості при перерозподілі процесорного часу мікроядро Neutrino2 використовує 3 алгоритми диспетчеризації для кожної черги потоків з однаковим пріоритетом:

- FIFO (обраний потік виконується, доки не передасть керування чи не перерветься більш пріоритетним потоком).

- Round Robin (обраний потік виконується, доки не передасть керування, не перерветься більш пріоритетним потоком або не закінчиться його інтервал часу).
- адаптивний алгоритм (якщо потік вичерпав виділений йому інтервал часу, то його пріоритет знижується на 1 за умови, що існує готовий до виконання потік з таким самим пріоритетом; якщо потік переходить у заблокований стан, то його пріоритет відновлюється до первісного значення).

Потік може встановити для себе будь-який з даних алгоритмів, однак він буде застосовуватися в тому випадку, якщо існує хоча б ще один потік з таким же пріоритетом і даним алгоритмом диспетчеризації. Перерозподіл часу процесора, відповідно до одного з приведених алгоритмів, порушується в тому випадку, якщо потік з великим пріоритетом стає готовим до виконання – у цьому випадку він негайно одержує керування, перериваючи виконання менш пріоритетного потоку.

## 2.4. Засоби синхронізації потоків

QNX-6 забезпечує POSIX-засоби синхронізації потоків, деякі з них можуть бути застосовні між потоками в різних процесах. Служби синхронізації вміщують наступне:

- м'ютекси,
- умовні змінні,
- бар'єри,
- sleeper-блокування,
- блокування читання/запису,
- семафори,
- FIFO-планування,
- передача/прийом/відповідь,
- атомарні операції.

Взаємно-виключні блокування, чи м'ютекси, є найбільш простими засобами синхронізації. М'ютекс використовується, щоб гарантувати монопольний доступ до загальнодоступних між потоками даних. Тільки один потік може блокувати м'ютекс у будь-який даний час. Потоки, що намагаються блокувати вже блокований м'ютекс, блокуються. Коли потік розблокує м'ютекс, потік з вищим пріоритетом, що очікує м'ютекс, розблокується і стане новим власником м'ютексу. Таким чином, потоки будуть послідовно захоплювати м'ютекс через критичний інтервал часу в порядку пріоритетної черги.

Умовна змінна, чи condvar, використовується, щоб блокувати потік у межах критичної секції програмного коду, доки певна умова не буде задоволена. Умова може бути довільною.

Бар'єр – це механізм синхронізації, що дозволяє призупинити декілька потоків, змушуючи їх чекати у визначеній точці (біля "бар'єру"), доки таких потоків не набереться певне число. Коли зазначене число таких потоків, що чекають у бар'єра, набралось, вони усі розблокуються і можуть продовжувати далі своє виконання.

Sleeper-блокування схожі на condvar, з декількома розходженнями. Подібно condvar, sleeper блокування може використовуватися, щоб блокувати потік, доки певна умова не буде задовільнена. На відміну від condvar, sleeper-блокування мають

"захований" м'ютекс, який програмно не доступний, і тому їх використання більш просте й надійне.

Блокування читання/запису використовуються для розв'язку проблеми "читач/письменник". Коли певний потік намагається записати дані до загальної області пам'яті, то за допомогою використання блокувань читання/запису його запит може бути відхилений, доки потік, що читає ці дані не розблокує її. Аналогічні дії можна виконати і при спробі читання під час запису даних іншим потоком.

Семафори аналогічні м'ютексам з тим розходженням, що доступ до ресурсу має не один потік, а кілька. Звільнення ресурсу збільшує семафор на одиницю, а захоплення – зменшує на одиницю. Якщо потік очікує на семафорі, що є позитивним, він не буде блокований. Чекання на не позитивному семафорі блокує, поки деякий інший потік не збільшує його. Семафори були визначені, щоб синхронізувати процеси. Для синхронізації між потоками в окремому процесі, м'ютекс буде більш ефективний, ніж семафори.

FIFO-алгоритм може гарантувати, що ніякі два потоки того ж самого пріоритету не виконають критичну секцію водночас. Усі FIFO-потоки в системі одного пріоритету будуть виконуватися по черзі, доки вони добровільно не звільнять процесор.

Передачі повідомлень Send/Receive/Reply здійснюють неявну синхронізацію за характером блокування. Це єдина синхронізація, що може використовуватися в мережі.

У деяких випадках необхідно виконати коротку операцію з гарантією, що операція не буде вивантажуватися іншим потоком чи програмою обробки переривання. Neutrino забезпечує атомарні операції для додавання значення, віднімання значення, очищення бітів, встановлення бітів.

## **2.5. Організація зв'язку між процесами**

Мікроядро Neutrino2 пропонує наступні форми IPC:

- передача повідомлень,
- сигнали,
- черги повідомлень POSIX,
- загальнодоступна пам'ять,
- канали,
- FIFOs.

Потік, що виконує MsgSend(), надсилаючи повідомлення до іншого потоку (який може бути в межах іншого процесу), буде блокований, доки той інший потік не виконає MsgReceive(), обробить повідомлення, і не виконає MsgReply(). Якщо потік виконує MsgReceive() без попередньо посланого повідомлення, то він блокується, доки інший потік не виконує MsgSend(), тобто не надішле повідомлення (рис. 3).

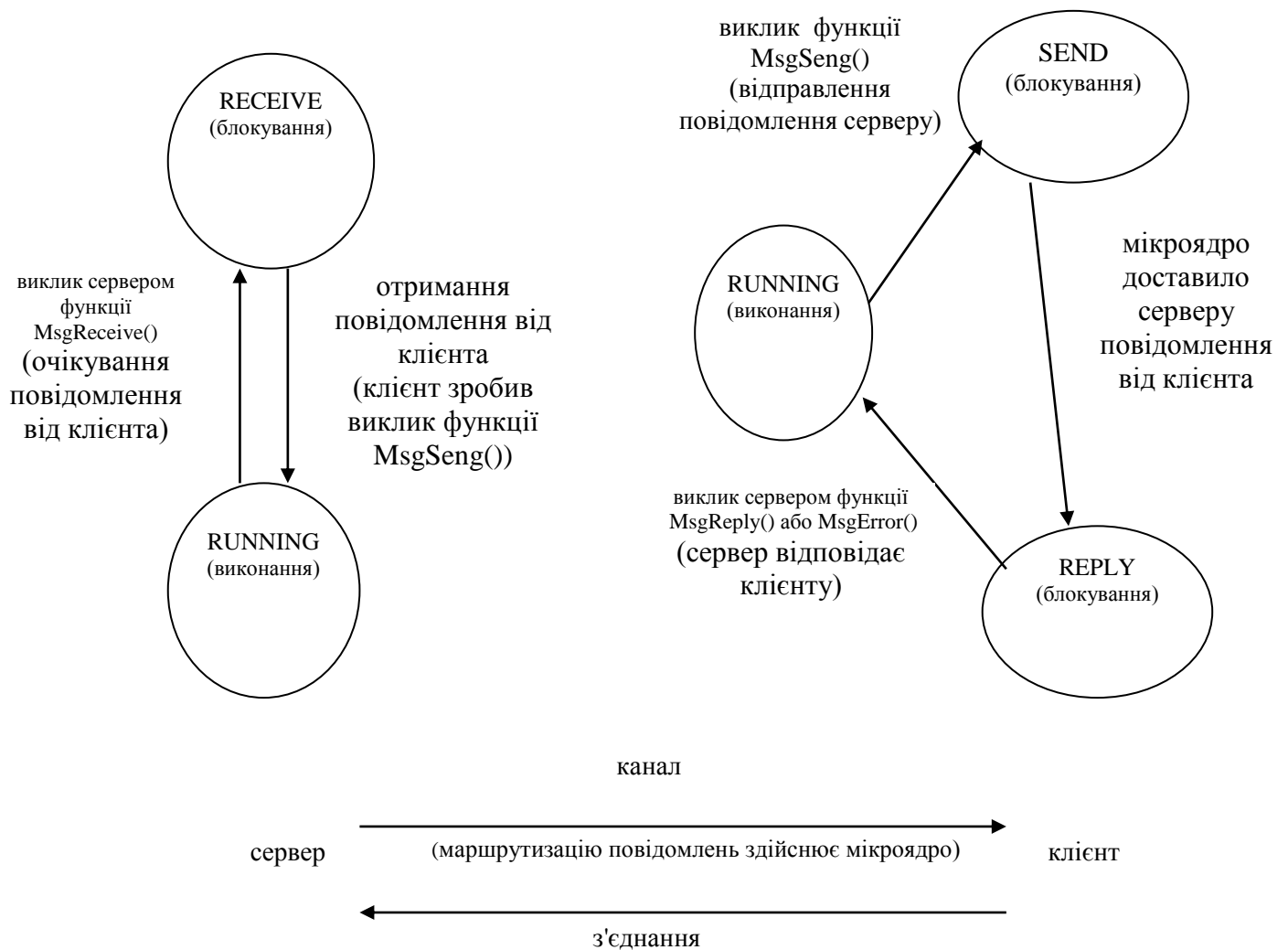


Рис. 3. Обмін повідомленнями між клієнтом та сервером

Це блокування синхронізує потік-відправник, що дозволяє серверу відповідати клієнту і продовжувати обробку, у той час як ядро асинхронно передає дані відповіді до потоку-відправника і позначають його як готовий до виконання. Функція `MsgReply()` використовується сервером для відповіді клієнту, щоб повернути нуль чи більшу кількість байтів, а функція `MsgError()` використовується, щоб повернути клієнту тільки стан. Обидві функції розблокують клієнта від його `MsgSend()`.

Служби передачі повідомлень копіюють повідомлення безпосереднє з адресного простору одного потоку до іншого без буферизації. Neutrino не аналізує зміст повідомлення – дані в повідомленні мають значення тільки для відправника й одержувача.

Для простих повідомлень Neutrino забезпечує функції, що беруть вказівник безпосередньо на буфер без потреби в IOV (Input-Output Vector – вектор введення/виведення).

У Neutrino передача повідомлень спрямована скоріше до каналів і підключень, ніж безпосередньо від потоку до потоку. Потік, що бажає одержати повідомлення,

спочатку створює канал; інший потік, що бажає послати повідомлення першому потоку, повинен спочатку створити підключення, приєднуючись до створеного каналу.

Як тільки підключення встановлені, клієнти можуть виконувати `MsgSend()`. Якщо багато потоків у процесі прикріплюються на той же самий канал, то одне підключення буде розподілено між усіма потоками. Канали і підключення в межах процесу мають цілочисловий ідентифікатор. Клієнтські підключення відображаються безпосередньо в дескриптори файлів.

Додатково до синхронних `Send/Receive/Reply` служб, Neutrino також підтримує передачу повідомлень фіксованого розміру без блокування його відправника. Вони визначені як імпульси і мають чотири байти даних плюс один байт коду. Імпульси часто використовуються як механізм повідомлення в межах програм обробки переривань. Вони також дозволяють серверам відповідати клієнтам без їхнього блокування.

Передача повідомлень API (Інтерфейс прикладної програми) складається з наступних функцій (табл. 3).

Таблиця 3

Функція	Опис
<code>MsgSend()</code>	Надіслати повідомлення і заблокуватися до отримання відповіді
<code>MsgReceive()</code>	Очікувати на повідомлення
<code>MsgReceivePulse()</code>	Очікувати на імпульс
<code>MsgReply()</code>	Відповісти на повідомлення
<code>MsgError()</code>	Відповісти тільки зі станом помилки.
<code>MsgRead()</code>	Читати додаткові дані від отриманого повідомлення
<code>MsgWrite()</code>	Записати додаткові дані до повідомлення відповіді
<code>MsgInfo()</code>	Одержати інформацію про отримане повідомлення
<code>MsgSendPulse()</code>	Надіслати імпульс
<code>MsgDeliverEvent()</code>	Надіслати клієнту подію
<code>MsgKeyData()</code>	<b>Повідомлення перевірки ключа</b>

Стандарт POSIX визначає набір неблокуючих засобів передачі повідомлень, відомих як черги повідомлень. На відміну від примітивів передачі повідомлень Neutrino2, черги повідомлень постійно знаходяться поза ядром. Черги повідомлень здійснюють проміжне накопичення повідомлень, вони існують незалежно від процесів, що їх використовують. В Neutrino2 створені черги повідомлення будуть з'являтися в просторі імен файлів у каталозі `/dev/mqueue`.

Загальнодоступна пам'ять пропонує процесам використовувати вказівники, щоб безпосередньо читати і записувати в неї. Загальнодоступна пам'ять часто використовується разом з одним із примітивів синхронізації, щоб зробити модифікації даних атомарними. Загальнодоступна пам'ять найбільш ефективна, коли використовується для модифікування великої кількості даних. Семафори і м'ютекси – придатні примітиви синхронізації для використання їх з



загальнодоступною пам'яттю. Загальнодоступна пам'ять реалізована в Neutrino у менеджері процесів (procnto).

Канал – це безіменний файл, що служить як канал введення/виведення між двома чи більше процесами, коли один процес записує в канал, а інший читає з каналу. Менеджер каналу забезпечує буферизацію даних. Канал буде вилучений, як тільки обоє його кінці закрилися. Канали звичайно використовуються, коли два процеси хочуть працювати паралельно з даними, що переміщуються від одного процесу до іншого в одному напрямку.

FIFOs – це ті ж самі канали, за винятком того, що вони є постійними файлами, які збережені в filesystem каталогах.

## 2.6. Таймери

Neutrino2 безпосередньо забезпечує повний набір функціональних можливостей таймерів.

POSIX-модель таймера забезпечує можливість встановити таймер на:

- абсолютну дату,
- відносну дату (в наносекундах від часу встановлення таймера),
- циклічну роботу (кожні *n* наносекунд).

Циклічний режим є дуже важливим, тому що використання таймерів може бути застосовано як періодичне джерело подій. Взагалі таймери – це джерело подій у Neutrino2.

Neutrino2 забезпечує функцію TimerTimeout(), що дозволяє додаткові визначати список станів блокування, щоб запустити очікування на зазначений час. Після закінчення часу чекання таймер буде заблокований і керування перейде додаткові. В табл.4 наведені функції Neutrino2 для роботи з таймерами.

Таблиця 4

Мікроядро	POSIX	Опис
TimerAlarm()	alarm()	Встановити тривогу процесу.
TimerCreate()	timer_create()	Створити інтервальний таймер.
TimerDestroy()	timer_delete()	Знищити інтервальний таймер.
TimerGettime()	timer_gettime()	Одержати час, що залишився на інтервальному таймері.
TimerSettime()	timer_settime()	Запустити інтервальний таймер.
TimerTimeout()	sleep(), nanosleep(), sigtimedwait(), pthread_cond_timedwait(), pthread_mutex_trylock(), intr_timed_wait()	Запускає очікування на зазначений час

## 2.7. Оброблювачі переривань

Neutrino цілком забезпечує керування апаратними перериваннями. Для обробки переривання додатку потрібно прив'язати необхідну функцію обробки до шуканого переривання, використовуючи виклики ядра. До одного апаратного переривання може бути прив'язано кілька оброблювачів, що одержують керування відповідно до пріоритетів потоків, яким вони належать.

Час очікування переривання – час від моменту появи апаратного переривання до виконання першої команди програми обробки переривання. Деякі критичні секції коду вимагають, щоб переривання були тимчасово заблоковані. Максимальний час блокування переривань звичайно визначає час очікування переривання в найгіршому випадку – у QNX цей час дуже малий і становить приблизно 0,5-2 мкс.

У деяких випадках, оброблювач апаратного переривання з низьким пріоритетом повинен очікувати виконання потоку з більш високим пріоритетом. Час очікування диспетчеризації – це час між останньою командою програми користувача і виконанням першої команди потоку драйвера. Він визначає час, який потрібен, щоб зберегти контекст активного потоку і відновити контекст потоку драйвера. Час очікування диспетчеризації більший за час очікування переривання, але він також дуже малий в QNX-6. Функції API для обробки переривань засновані на системних запитах ядра (табл.5).

Таблиця 5

Функція	Опис
InterruptAttach ()	Прикріпити функцію до вектора переривання
InterruptAttachEvent()	Генерувати подію переривання. Ніяка програма обробки переривання користувача не виконується.
InterruptDetach ()	Відокремити функцію від переривання, використовуючи ID, повернутий InterruptAttach() чи InterruptAttachEvent ()
InterruptWait ()	Очікування переривання
InterruptEnable ()	Включити апаратні переривання
InterruptDisable ()	Відключити апаратні переривання
InterruptMask ()	Маскувати апаратне переривання
InterruptUnmask ()	Демаскувати апаратне переривання
InterruptLock ()	Блокувати критичну секцію коду між програмою обробки переривання і потоком.
InterruptUnlock ()	Видалити блокування на критичній секції коду

Використовуючи цей API, привілейований потік рівня користувача може викликати InterruptAttach() чи InterruptAttachEvent(). ОС QNX-6 дозволяє багатьом оброблювачам переривання обробляти кожен номер апаратного переривання.

### 3. Розробка консольних додатків

Розглянемо процес створення консольних додатків на прикладі двох програм, які виконують обмін повідомленнями поміж собою. Одна з них ("клієнт") посилає повідомлення іншій програмі ("серверу"), яка після отримання повідомлення відповідає клієнтові. Програми працюють як консольні додатки, причому після відповіді на своє повідомлення програма-клієнт завершується, а програма-сервер ніколи не завершується, бо продовжує чекати на інші повідомлення.

Будемо вважати, що на комп'ютері завантажена ОС QNX-6 та графічна оболонка Photon. Для створення тексту програми-сервера необхідно послідовно вибрати пункти меню Applications, Editor, File, New та набрати наступний текст програми:

```
#include <stdio.h>
#include <sys/neutrino.h>
#include <fcntl.h>
#include <stdlib.h>
#include <unistd.h>

int main()
{
    int rcvid;
    int chid;
    int pid;
    int fd;
    char message [512];
    pid=getpid();
    chid=ChannelCreate(0);

    printf("\nI am Server PID=%d . I create Channel %d\n",pid,chid);
    fflush(stdout);

    fd = open("server.msg", O_WRONLY|O_CREAT);
    if(fd==-1)printf("\nopen error = %d\n",fd);
        else printf("I was written my PID in file 'server.msg' Ok!\n");
    write (fd,&pid,sizeof(pid));
    close (fd);

    while(1)
    { rcvid=MsgReceive(chid, message, sizeof(message), NULL);
      printf("Recive message, rcvid %x\n:", rcvid);
      printf("Message body: %s\n", message);
      strcpy(message, "This is answer");
      MsgReply(rcvid, 0, message, sizeof(message));
    }
    return(0) ;
}
```

Після набору цього тексту його необхідно зберегти в файлі з іменем "server.c" в папці "/progs".

Для компіляції поданого тексту програми "server.c" необхідно відкрити псевдо термінал, натиснувши на кнопку "Terminal". Після цього з'явиться вікно терміналу, в якому треба набрати команду:

```
gcc /progs/server.c -o /progs/server.exe
```

Після успішного виконання зазначеної команди в папці "/progs" з'явиться файл з іменем "server.exe", який можна завантажувати на виконання.

Для створення тексту програми-клієнта можна відкрити новий редактор тексту, для чого необхідно послідовно вибрати пункти меню Applications, Editor, File, New, або можна скористатися вже відкритим редактором. Далі слід набрати наступний текст програми:

```
#include <stdio.h>
#include <sys/neutrino.h>
#include <fcntl.h>
#include <unistd.h>
char *smsg="This is buffer";
char rmsg[200];
int coid;

int main()
{
int pid_client;
int fd;
int pid_ser;
char* filename="server.msg";
pid_client=getpid();
printf("\nI am Client %d .",pid_client);
printf("Testing of message exchanging\n");

fd=open(filename,O_RDONLY);
if (fd==-1) printf("\nopen file %s error %d",filename,fd);
    else printf("file %s opennig Ok!",filename);
read(fd,&pid_ser,sizeof(pid_ser));
close(fd);

printf("\nPID servera %d .",pid_ser);
fflush(stdout);

coid=ConnectAttach(0,pid_ser,1,0,0);
if (coid==-1) {printf("Connection error\n");exit(1);}
if (MsgSend(coid,smsg,strlen(smsg)+1,rmsg,sizeof(rmsg))==-1)
{printf("Send message error\n");exit(1);}
if (strlen(rmsg)>0) {printf("Process return %s\n",rmsg);}
return(0);
}
```

Після набору цього тексту його необхідно зберегти в файлі з іменем "client.c" в папці "/progs".

Для компіляції поданого тексту програми "client.c" необхідно відкрити новий псевдо термінал, натиснувши на кнопку "Terminal". Після того, як з'явиться вікно терміналу, необхідно набрати команду:

```
gcc /progs/client.c -o /progs/client.exe
```

Після успішного виконання зазначеної команди в папці "/progs" з'явиться файл з іменем "client.exe", який можна завантажувати на виконання.

Після розробки завантажувальних файлів необхідно на одному терміналі завантажити файл "server.exe", а на іншому терміналі – файл "client.exe". Після їх завантаження на екранах обох терміналів з'являться відповідні повідомлення, які вказують на результати обміну повідомленнями між клієнтом і сервером. Зауважимо, що програму-клієнт можна завантажувати декілька разів, спостерігаючи на терміналах відповідні результати після її запуску, а програма-сервер завантажується тільки один раз та захоплює "свій" термінал. Для того, щоб програму-сервер закінчити, необхідно або закрити відповідний до неї термінал, або з будь-якого іншого терміналу подати команду ps, визначити pid програми server.exe та подати команду kill <pid>.

### **Контрольні питання**

1. Скільки потоків може мати кожен процес?
2. Чи може процес породити процес? Чи може процес породити потік?
3. Чи може потік породити процес? Чи може потік породити потік?
4. Як виконується копіювання (дублювання) процесів?
5. Яка програмна одиниця підлягає плануванню та диспетчеризації: процес чи потік?
6. Якими ресурсами володіє процес та якими ресурсами володіє потік?
7. Яким чином ідентифікується процес в системі? Чи можна ідентифікувати потік процесу?
8. Якщо завершується головний потік процесу, то чи будуть "автоматично" завершені інші потоки цього ж процесу?
9. В яких станах може знаходитися потік?
10. Що позначає стан потоку NANOSLEEP? Чи є виклик функції sleep() системним викликом?
11. Які об'єкти підтримує мікроядро Neutrino2?
12. Які об'єкти синхронізації підтримує мікроядро Neutrino2?
13. З якою метою використовується мютекс? Чи може мютекс бути розподілений поміж потоками? А поміж процесами?
14. Чим відрізняється семафор від мютекса?
15. Що позначає засіб синхронізації "бар'єр"?
16. В чому відмінність засобу синхронізації "sleepon" (чекаючі блокування) від "condvars" (умовні змінні)?
17. Яким чином один потік може чекати завершення роботи іншого потоку?
18. Яка різниця між FIFO- та RR-диспетчеризаціями?
19. Скільки пріоритетів можна назначити в системі реального часу на базі QNX-6?
20. Подайте схему обміну повідомленнями поміж двома процесами.
21. Чи може потік направити повідомлення сам собі?
22. Наведіть основні функції, які можна використати для обміну повідомленнями.

## Завдання до лабораторної роботи

### Завдання 1

Скласти однопотокову програму, що друкує свої атрибути (пріоритет, pid, дисципліну диспетчеризації) та час свого завантаження на виконання, а потім наказує командному інтерпретаторові виконати команду:

1) ps, 2) ls, 3) cd, 4) pwd, 5) ab, 6) mount/dev/fd0 fs/floppy, 7) pterm, 8) pidin, 9) gcc, 10) mqc, 11) pwd 12) завантажити себе ще раз, причому нова завантажена програма вже не може завантажувати сама себе.

Вказати, що позначають зазначені стандартні команди і які опції вони мають.

### Завдання 2

Скласти багатопотокову програму, в якій кожен потік інформує про свої атрибути (пріоритет і дисципліну диспетчеризації) та час свого завантаження на виконання, а потім "засинає" на декілька секунд, після чого друкує інформацію про поточний час і завершується. Час, на який повинен "засинати" потік, передається потоку як аргумент виклику. Потоки повинні бути уособленими (detached). Кількість потоків та атрибути потоків взяти з таблиці варіантів. Пояснити послідовність надрукованих на екрані даних.

Варіант завдання	Кількість потоків (без головного)	Атрибути потоку 1	Атрибути потоку 2	Атрибути потоку 3
1	2	RR,10	RR,12	-
2	3	FIFO,10	успадковає	FIFO,35
3	3	успадковає	FIFO,12	RR,23
4	3	FIFO,25	успадковає	FIFO,11
5	2	RR,11	успадковає	-
6	2	RR,15	RR,15	-
7	2	FIFO,20	FIFO,10	-
8	3	успадковає	FIFO,17	RR,15
9	1	FIFO,18	-	-
10	3	RR,60	FIFO,11	успадковає
11	2	RR,21	успадковає	-
12	3	успадковає	RR,10	RR,14
13	3	FIFO,14	FIFO,7	успадковає
14	2	успадковає	FIFO,15	-
15	2	RR,50	успадковає	-
16	3	RR,1	FIFO,12	FIFO,29
17	1	FIFO,16	-	-
18	1	FIFO,22	-	-
19	3	FIFO,45	RR,8	FIFO,55
20	2	успадковає	FIFO,16	-
21	2	RR,5	успадковає	-
22	3	успадковає	успадковає	FIFO,15
23	3	FIFO,20	RR,9	FIFO,12
24	2	успадковає	FIFO,17	-
25	1	RR,7	-	-
26	1	RR,3	-	-
27	2	RR,30	FIFO,11	-
28	2	FIFO,12	успадковає	-

### **Завдання 3**

Виконати розробку програмного коду однієї з задач.

**Задача 1.** Розробити схеми взаємодії програмних компонентів для реалізації такого фрагменту СРЧ. Клієнт кожні 15 секунд запитує у сервера дані про температуру та тиск певного фізичного процесу. У відповідь на запит, сервер створює та запускає на виконання два потоки для вимірювання значень вказаних параметрів, після чого відправляє ці значення у повідомленні клієнту, а потім знищує створені потоки, чекаючи на наступну команду клієнта, але не більше, ніж як 2 хвилини. Якщо через 2 хвилини сервер не отримує запит від клієнта, він завершує свою роботу. Вказівки: сервер не може відповідати клієнту неповним набором даних (тобто тільки значенням температури, або тільки значенням тиску); для синхронізації потоків сервера використати метод "бар'єр" (barrier).

**Задача 2.** Розробити схеми взаємодії програмних компонентів для реалізації такого фрагменту СРЧ. Клієнт кожні 20 секунд запитує у сервера дані про вологість, температуру та тиск повітря. У відповідь на кожен запит, сервер створює та запускає на виконання потоки для вимірювання значень вказаних параметрів, після чого відправляє ці значення у повідомленні клієнту, а потім знищує створені потоки, чекаючи на наступну команду клієнта. Після 10 запитів клієнт завершує роботу, а сервер ніколи не завершує свою роботу. Вказівки: сервер не може відповідати клієнту неповним набором даних; для синхронізації потоків сервера слід використати метод "приєднання" (joining).

**Задача 3.** Розробити схеми взаємодії програмних компонентів для реалізації такого фрагменту СРЧ. З клавіатури вводиться текст з літер та цифр. Програма (головний потік) створює та запускає на виконання три потоки, де перший потік ("письменник") запам'ятовує цей текст в буферній області пам'яті, а другий та третій потоки ("читачі") зчитують з цієї області пам'яті, причому другий потік зчитує тільки літери, а третій потік зчитує тільки цифри. Після цього кожен потік-"читач" друкує відповідні символи на екрані та звільняє від них буферну область пам'яті. Через 5 хвилин після початку своєї роботи потік-"письменник" завершує свою роботу та повідомляє про це головний потік програми. Після цього повідомлення головний потік програми дає наказ потокам-"читачам" завершити своє виконання, і після їх завершення завершується сам. Вказівки: потоки програми повинні бути синхронизовані для розподілу загального ресурсу (буферної області пам'яті) за методом блокування "читач-письменник" (reader-writer lock); робота потоків-"читачів" повинна бути організована за методом FIFO-диспетчерізації з використанням функції sched\_yield().

**Задача 4.** Розробити схеми взаємодії програмних компонентів для реалізації такого фрагменту СРЧ. З клавіатури вводиться текст. Програма (головний потік) створює та запускає на виконання три потоки, де перший потік запам'ятовує цей текст в буферній області пам'яті, а другий та третій потоки зчитують текст з цієї області пам'яті словами (слова розділяються символом "пропуск") та друкують їх на екрані, звільняючи від них буферну область пам'яті. Через 7 хвилин після початку своєї роботи потік-"письменник" завершує свою роботу та повідомляє про це потоки-"читачі". Після цього повідомлення потоки-"читачі" завершують свою роботу та повідомляють про це головний потік програми, який після цього завершується сам. Вказівки: потоки програми повинні бути синхронизовані для розподілу загального ресурсу (буферної області пам'яті) за методом блокування

"читач-письменник" (reader-writer lock); робота потоків-"читачів" повинна бути організована за методом карусельної (Round Robin) диспетчеризації.

**Задача 5.** Розробити схеми взаємодії програмних компонентів для реалізації такого фрагменту СРЧ. Сервер збирає дані від своїх 5 субсерверів та відповідає клієнту на запит про значення певного загального показника. Як приклад, використати таку інтепретацію: субсервер – це інтелектуальний лічильник електрики, сервер – це центральний пульт з підрахунку кількості спожитої електрики, клієнт – це керівник підприємства. Вказівки: субсервери повинні бути потоками сервера; час відповіді на запит клієнта не повинен перевищувати 100 мілісекунд.

**Задача 6.** Розробити схеми взаємодії програмних компонентів для реалізації такого фрагменту СРЧ. Сервер збирає дані від своїх субсерверів та відповідає клієнту на запит про значення певного загального показника. Як приклад, використати таку інтепретацію: субсервер – це інтелектуальний лічильник газу, сервер – це центральний пульт з підрахунку кількості спожитого газу, клієнт – це керівник підприємства. Вказівки: субсервери повинні бути окремими процесами (програмами).

**Задача 7.** Розробити схеми взаємодії програмних компонентів для реалізації такого фрагменту СРЧ. За командою клієнта, сервер запускає свій субсервер, який періодичними імпульсами кожні 2 секунди повідомляє сервер про свою працездатність та передає при цьому певні дані у вигляді послідовності з 2 цілих двобайтових чисел. Нехай через 100 секунд після початку своєї роботи субсервер блокується з невідомої причини. В цьому випадку сервер не отримає чергового імпульсу від свого субсервера і, тому, повинен доповісти клієнту про свою неможливість продовжувати виконання його команди.

**Задача 8.** Розробити схеми взаємодії програмних компонентів для реалізації такого фрагменту СРЧ. Сервер за командою клієнта запускає певний потік (субсервер), який чекає, коли настане заданий момент часу (наприклад, 10:00:00 15.10.2003), після чого формує загальний сигнал на розблокування всім потокам сервера, що були блоковані попередньою командою клієнта.

**Задача 9.** Розробити схеми взаємодії програмних компонентів для реалізації такого фрагменту СРЧ. Один потік за перериваннями таймеру зчитує байт даних з порту 0x03F8 (або випадково формує байт даних) та записує його до буферної області пам'яті. Другий потік перевіряє наявність в зазначеній пам'яті нових даних і, якщо вони є, зображує їх у вигляді нових точок певного графіку. Третій потік сканує клавіатуру і, якщо натиснута клавіша "Esc", завершує роботу програми. Вказівки: для синхронізації потоків використати метод "чекаючого блокування" (sleeps).

**Задача 10.** Розробити схеми взаємодії програмних компонентів для реалізації такого фрагменту СРЧ. Інформація про ціни на продукцію записується п'ятьма виробниками кожного виду продукції до бази даних (файлу). Одна з програм – сервер – стежить за змінами в базі даних і, якщо вони з'являються, одразу ж розсилає нову інформацію про ціну трьом іншим зацікавленим клієнтам. Вказівки: при моделюванні моменти часу зміни цін задавати періодично з періодом 20, 21, 22, 23 та 24 секунди відповідно для кожного виробника.

**Задача 11.** Розробити схеми взаємодії та частини програмного коду, що реалізують такий фрагмент СРЧ. Інформація про ціни на продукцію записується трьома виробниками кожного виду продукції до бази даних (файлу). Одна з



програм – сервер – стежить за змінами в базі даних і, якщо вони з'являються, одразу ж генерує "імпульс" чотирьом іншим зацікавленим клієнтам, які, отримавши цей "імпульс", одразу ж починають читати нову інформацію. Вказівки: при моделюванні моменти часу зміни цін задавати періодично з періодом 20, 21 та 22 секунди відповідно для кожного виробника.

**Задача 12.** Розробити схеми взаємодії програмних компонентів для реалізації такого фрагменту СРЧ. Клієнт дає завдання серверу монополізувати доступ до певного ресурсу (області пам'яті, СОМ-порту і т.і.), який синхронізується за методом "взаємного виключення" (mutex). Якщо сервер встигає це зробити на протязі 5 секунд, то він повідомляє клієнту про виконання його завдання, інакше повідомляє про неможливість зайняти ресурс.

**Задача 13.** Виконати моделювання процесу продажу 42 авіаквитків на рейс 777 авіакомпанії ROOT, якщо є 7 авіакаса, які запитують сервер продажу авіаквитків кожні 1,2,3,4,5,6 та 7 секунд відповідно та продають їх, причому сьома авіакаса більш пріоритетна, ніж шоста, а шоста більш пріоритетна, ніж всі інші, які мають однаковий пріоритет. Кожні 5 секунд в восьмій авіакасі, яка є самою пріоритетною, виконується повернення одного проданого авіаквитка. Програма завершується, коли всі авіаквитки є проданими. Вказівки: потоки програми повинні бути синхронизовані для розподілу загального ресурсу (авіаквитки) за методом блокування "читач-письменник" (reader-writer lock); робота потоків повинна бути організована за методом карусельної (Round Robin) диспетчеризації.

**Задача 14.** Розробити схеми взаємодії програмних компонентів для реалізації такого фрагменту СРЧ. Три потоки сервера читають текстовий файл, розбираючи його вміст по словам. По закінченні читання файлу сервер повідомляє клієнту, що файл прочитано. Після цього клієнт подає серверу команду на знищення зазначених трьох потоків та завершення роботи сервера. Через 1 секунду клієнт перевіряє, що його команду виконано.

**Задача 15.** Розробити схеми взаємодії програмних компонентів для реалізації такого фрагменту СРЧ. Сервер за командами клієнта малює на своєму екрані прості графічні зображення (прямокутники, кола, трикутники, лінії і т.і.) в заданій позиції заданого кольору заданого розміру. Якщо потрібного образу не існує, то сервер відповідає клієнту відмовою, інакше – підтвердженням команди. Час, який сервер може витратити на відповідь, не повинен перевищувати половину секунди.

**Задача 16.** Розробити схеми взаємодії програмних компонентів для реалізації такого фрагменту СРЧ. Сервер приймає запити від 5 клієнтських програм і завжди відповідає рівно через 0,5 секунди після отримання запиту певним набором даних, які очікує клієнт. Клієнтські програми генерують запити з частотою 5 секунд. Через 200 секунд після відповіді на перший запит сервер завершує свою роботу та повідомляє про це всім клієнтам.

**Задача 17.** Розробити схеми взаємодії програмних компонентів для реалізації такого фрагменту СРЧ. П'ять потоків сервера обслуговують 5 клієнтських програм, причому перший потік обслуговує першого клієнта, другий потік – другого і т.д. Запит другого клієнта не можна обслуговувати, якщо є запит першого клієнта, а запити третього, четвертого та п'ятого клієнтів рівні за своїм пріоритетом, але їх не можна обслуговувати, якщо є запит другого клієнта. Запити першого клієнта виникають кожні 10 секунд, другого – кожні 8 секунд, а інших – кожні 5 секунд. Сервер завершує свою роботу рівно через 300 секунд після початку своєї роботи.

**Задача 18.** Розробити схеми взаємодії програмних компонентів для реалізації такого фрагменту СРЧ. Сервер є "розповсюджувачем" рекламної інформації (наприклад, про ціну певного товару). та відповідає на запити клієнтів одним й тим же повідомленням. Нехай одночасно 10 клієнтів роблять запит на цю інформацію. Тоді сервер запускає пул потоків, які відповідають клієнтам. Вказівки: сервер одночасно може організувати пул потоків розміром від 2 до 7 потоків.

**Задача 19.** Розробити схеми взаємодії програмних компонентів для реалізації такого фрагменту СРЧ. Десять клієнтських програм одночасно підтримують зв'язок з сервером цін на продукцію. Якщо ціна на продукцію змінюється, то сервер одразу ж повідомляє про це клієнтам. Крім того, ціна на продукцію контролюється клієнтами один раз на добу о 9 годині. Вказівки: сьогодні в 9:05 ціна на продукцію була змінена, і сервер повинен повідомити про це клієнтам. Сервер ніколи не завершується, а його параметри (дескриптор вузла, дескриптор процесу та дескриптор каналу) записані до файлу `"/root/price_list"`.

**Задача 20.** Розробити схеми взаємодії програмних компонентів для реалізації такого фрагменту СРЧ. Інформація про ціни на продукцію записується п'ятьма виробниками кожного виду продукції до бази даних (файлу). Одна з програм – сервер – стежить за змінами в базі даних і, якщо вони з'являються, одразу ж розсилає нову інформацію про ціну трьом іншим зацікавленим клієнтам. Вказівки: при моделюванні моменти часу зміни цін задавати періодично з періодом 20, 21, 22, 23 та 24 секунди відповідно для кожного виробника.

**Задача 21.** Розробити схеми взаємодії та частини програмного коду, що реалізують такий фрагмент СРЧ. Інформація про ціни на продукцію записується трьома виробниками кожного виду продукції до бази даних (файлу). Одна з програм – сервер – стежить за змінами в базі даних і, якщо вони з'являються, одразу ж генерує "імпульс" чотирьом іншим зацікавленим клієнтам, які, отримавши цей "імпульс", одразу ж починають читати нову інформацію. Вказівки: при моделюванні моменти часу зміни цін задавати періодично з періодом 20, 21 та 22 секунди відповідно для кожного виробника.

**Задача 22.** Розробити схеми взаємодії програмних компонентів для реалізації такого фрагменту СРЧ. Клієнт передає серверу графічне зображення; сервер зображує передане йому зображення на своєму екрані та "імпульсом" повідомляє клієнта про свою готовність до отримання наступного зображення. Вказівки: розмір зображення 200x200 пікселів; клієнт формує чергове зображення і передає його тільки тоді, коли він отримує "імпульс" від сервера.

**Задача 23.** Розробити схеми взаємодії програмних компонентів для реалізації такого фрагменту СРЧ. Клієнт передає серверу графічне зображення, а сервер зображує його на своєму екрані. Клієнт формує чергове зображення кожну секунду і одразу ж передає його серверу; сервер запускає два потоки, один з яких стежить за надходженням повідомлення від клієнта, а другий виконує безпосередньо функцію відображення на екран. Якщо другий потік сервера не встигає зобразити чергове зображення до надходження нового, то він залишає свою роботу і починає зображати нове зображення. Вказівки: розмір зображення 200x200 пікселів; визначити період роботи клієнта, коли сервер ще встигає повністю зображати кожне передане йому зображення вказаного розміру.

**Задача 24.** Розробити схеми взаємодії програмних компонентів для реалізації такого фрагменту СРЧ. П'ять клієнтів одночасно та періодично з періодом в 1

секунду передають певні графічні зображення серверу у вигляді повідомлень. Сервер запускає пул потоків та відображає ці зображення одночасно в п'яти вікнах. Вказівки: розмір кожного зображення 100x100 пікселів.

**Задача 25.** Розробити схеми взаємодії програмних компонентів для реалізації такого фрагменту СРЧ. Сервер підтримує діалог двох клієнтів, переправляючи їх повідомлення один одному; повідомлення можуть мати текстовий або графічний формат. Діалог продовжується рівно 1 хвилину.

**Задача 26.** Розробити схеми взаємодії програмних компонентів для реалізації такого фрагменту СРЧ. Сервер підтримує "нараду" трьох клієнтів, забезпечуючи обмін текстовими повідомленнями між ними. Сьогодні початок наради о 10:00, а закінчення – о 10:01. Нарада проходить за правилами: 1) передається одне слово кожного клієнта двом іншим клієнтам, 2) передача чергового слова виконується кожну секунду після початку наради; 3) після слова першого клієнта має "сказати" слово другий клієнт, після нього – третій клієнт, а потім знову перший і т.д.; 4) якщо у клієнта закінчилися "слова", то він повідомляє про це сервер за допомогою імпульсу, і сервер "надає слово" іншому клієнту; 5) якщо скінчилися слова у всіх клієнтів, то сервер передчасно перериває їх нараду, повідомляючи їх про це. Вказівки: слова кожного клієнта знаходяться у відповідних текстових файлах; сервер ніколи не закінчує свою роботу.

**Задача 27.** Розробити схеми взаємодії програмних компонентів для реалізації такого фрагменту СРЧ. Сервер підтримує "нараду" трьох клієнтів, забезпечуючи обмін текстовими повідомленнями між ними. Сьогодні початок наради о 10:00, а закінчення – о 10:01. Нарада проходить за правилами: 1) передається одне речення (послідовність слів, що закінчується крапкою) кожного клієнта двом іншим клієнтам, 2) після речення першого клієнта має "сказати" речення другий клієнт, після нього – третій клієнт, а потім знову перший і т.д.; 3) передача чергового речення виконується кожні 2 секунди після початку наради; 4) якщо клієнту немає чого "сказати" (наприклад, закінчився текст його "промови"), то він повідомляє про це сервер за допомогою імпульсу, і сервер "надає слово" іншому клієнту; 5) якщо скінчилися речення у всіх клієнтів, то сервер передчасно перериває їх нараду, повідомляючи їх про це. Вказівки: тексти речень кожного клієнта знаходяться у відповідних файлах; сервер закінчує свою роботу о 10:01, повідомляючи про це клієнтів, якщо вони ще не закінчили своєї роботи.

**Задача 28.** Розробити схеми взаємодії програмних компонентів для реалізації такого фрагменту СРЧ. Сервер підтримує "нараду" трьох клієнтів, забезпечуючи обмін текстовими повідомленнями між ними. Нарада проходить за правилами: 1) кожну секунду передається тільки одне речення (послідовність слів, що закінчується крапкою) клієнта двом іншим клієнтам; 2) після першого клієнта має "сказати" другий клієнт, після нього – третій клієнт, а потім знову перший і т.д.; 3) якщо клієнт "сказав" більше, ніж 100 слів, то сервер відключає цього клієнта від наради, повідомляючи його про це; 4) якщо клієнту немає чого "сказати" (наприклад, закінчився текст його "промови"), то він повідомляє про це сервер за допомогою імпульсу, і сервер "надає слово" іншому клієнту; 5) якщо скінчилися речення у всіх клієнтів, то сервер передчасно перериває їх нараду, повідомляючи їх про це. Вказівки: тексти речень кожного клієнта знаходяться у відповідних файлах; сервер ніколи не закінчує свою роботу.

## ЛАБОРАТОРНА РОБОТА №6

### Тема: Мікроядро Photon та інтегроване середовище Application Builder

**Мета:** Вивчення особливостей проектування та освоєння спеціальних засобів розробки графічного інтерфейсу систем реального часу та їх компонентів в інтегрованому середовищі Application Builder (AB) на базі мікроядра Photon. Знайомство з меню та командами середовища, робота з вікнами, палітрою компонентів та бібліотекою середовища Application Builder. Складання простих програм, компіляція програм та завантаження програм на виконання. Програмування графічних додатків систем реального часу на базі мікроядра Photon.

#### 1. Загальні відомості про мікроядро Photon

##### 1.1. Віконна система Photon microGUI

Віконна система Photon microGUI (GUI – Graphical User Interface) побудована за класичною схемою клієнт/сервер, але відрізняється від інших подібних систем, в першу чергу, обмеженою функціональністю безпосередньо графічного мікроядра Photon, коли більша частина функцій GUI розподіляється між взаємодіючими (на основі механізму IPC – Inter-Process Communication) процесами. Тому графічне мікроядро Photon потребує незначного об'єму пам'яті, а гнучка нарощувана архітектура дозволяє легко розширити можливості графічного інтерфейсу. Завдяки можливостям крос-платформеного зв'язку, графічні додатки Photon можуть бути використані практично в будь-якому середовищі на великій кількості апаратних платформ.

Мікроядро Photon виконується як маленький процес, що реалізує тільки декілька фундаментальних примітивів, які використовуються зовнішніми процесами для побудови віконної системи більш високого рівня функціональності. Для самого мікроядра Photon графічного "вікна" не існує. Мікроядро Photon не може також "малювати" що-небудь або керувати мишкою чи клавіатурою. Проте мікроядро Photon керує графічним середовищем.

Для керування середовищем GUI, Photon створює 3-вимірний "простір подій", де оперує регіонами, виконуючи відсікання та направлення різних подій за темпом їх проходження регіонами в цьому просторі подій. Така поведінка є аналогічною до поведінки мікроядра Neutrino операційної системи QNX-6, яке, наприклад, не підтримує функції введення/виведення для пристроїв або файлової системи, бо зазначені дії виконують зовнішні процеси, які надають ці послуги на високому (абстрактному) рівні. Це забезпечує можливість масштабування ОС та GUI за розміром програмного коду та функціональністю. Так, для систем с обмеженими ресурсами Photon може масштабуватися "вниз" за рахунок видалення процесів, які надають сервісні функції. Але Photon може масштабуватися і "вгору" до повнофункціональної системи за рахунок додавання процесів, які надають відповідні сервісні функції.

## 1.2. Простір подій Photon

Оснoву для роботи мікроядра Photon складає уявлений простір подій, в який інші процеси можуть поміщати регіони. Використовуючи QNX IPC для зв'язку з мікроядром Photon, ці процеси керують своїми регіонами для надання певних графічних сервісних функцій високого рівня або для виконання графічних функцій додатків користувача.

"Простір подій" можна представити як порожній тривимірний простір з "кореневим регіоном" на задньому плані. Користувачі нібито "дивляться всередину" цього простору подій. Додатки розміщують регіони в тривимірний простір між кореневим регіоном та користувачем; вони використовують ці регіони для генерації та прийому різних типів подій в цьому просторі. Процеси, які виконують обслуговування драйверів пристроїв, розміщують регіони на передній план простору подій. Крім керування простором подій та кореневим регіоном, мікроядро Photon підтримує екранний вказівник (курсор), який проектується як подія рисування з напрямком до користувача.

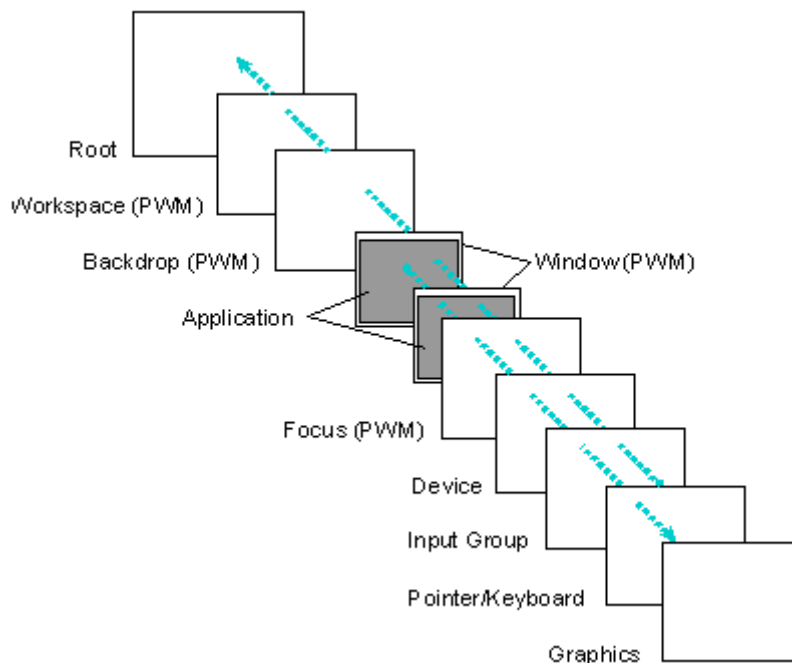


Рис.1. Простір подій

Photon використовує послідовність регіонів, починаючи від кореневого регіону на задньому плані простору подій до графічного регіону спереду. Події рисування пересуваються від регіонів додатків до графічного регіону. Події введення виникають в регіоні курсору/клавіатури і пересуваються за напрямком до кореневого регіону.

Події, які пересуваються в просторі подій, можна представити собі як "фотони" (це і дало назву віконній системі). Самі події складаються з набору прямокутних областей та прикріплених до них даних. Під час пересування подій в просторі подій їх прямокутники пересікають регіони, які належать різним процесам (додаткам).

Про події, які пересуваються від кореневого регіону, кажуть, що вони пересуваються зовні (за напрямком до користувача), в той час як про події, які пересуваються від користувача, кажуть, що вони пересуваються всередину, за напрямком до кореневого регіону.

Взаємодія між подіями та регіонами є основою введення/введення в Photon. Події мишки, клавіатури та світлового пера пересуваються від користувача до кореневого регіону, з "прикріпленим" до них положенням курсору. Події рисування виникають в регіонах й пересуваються за напрямком до регіону пристрою та користувача.

### 1.3. Регіони

Кожному регіону відповідає прямокутна область, яка визначає його положення в 3-вимірному просторі подій. Регіон також має атрибути, які визначають, як він взаємодіє з різними класами подій при пересіканні ними регіону. Взаємодія регіону з подіями визначається двома бітовими масками: маскою чутливості та маскою непрозорості.

Маска чутливості визначає, чи повинен процес-власник регіону сповіщати про пересікання регіону тією чи іншою подією. Кожен біт маски чутливості визначає, чи є чутливим регіон до визначеного типу подій. Коли подія пересікає регіон, для якого встановлений біт (дорівнює 1), копія цієї події ставиться в чергу процес-власника регіону, повідомляючи додаток про проходження події через регіон. Таке повідомлення ніяк не змінює саму подію.

Маска непрозорості визначає прозорість регіону для тих чи інших подій. Кожен біт цієї маски визначає, чи є регіон прозорим для визначеного типу події. При проходженні події крізь "непрозорий" регіон вона модифікується.

Зазначені бітові маски можуть бути спільно використані для досягнення різних результатів (табл.1).

Таблиця 1

Сполучення бітових масок

Сполучення бітових масок	Опис
Нечутливий, прозорий	При проходженні події через регіон вона не модифікується і власник регіону не повідомляється. Процес-власник регіону просто не цікавиться подією.
Нечутливий, непрозорий	При проходженні події через регіон вона відсікається; власник регіону не повідомляється. Більшість додатків використовують таку комбінацію атрибутів для відсікання подій рисування, щоб уникнути перерисування вікна подіями рисування, які йдуть від вікон, що лежать під ним.
Чутливий, прозорий	Копія події направляється власнику регіону; подія продовжує пересування в просторі подій, не змінюючись. Процес, який бажає реєструвати проходження всіх подій, може використовувати таку комбінацію.
Чутливий, непрозорий	Копія події направляється власнику регіону; подія відсікається регіоном. Встановивши таку комбінацію масок, подія може грати роль фільтра або перетворювача. Додаток може обробити будь-яку отриману подію, регенерувати її та при необхідності перетворити її при цьому певним чином, можливо, змінивши напрямок пересування або координати. Наприклад, регіон може поглинати події світлового пера, виконувати розпізнавання почерку, а потім генерувати еквівалентні події натиснення клавіш.

#### 1.4. Події

Подібно до регіонів, події можуть відноситись до різних класів та мати різні атрибути, такі як регіон походження, тип, напрямок, прикріплений список прямокутників, специфічні для даної події дані.

На одміну від більшості віконних систем, Photon класифікує в якості подій не лише введення (перо, мишка, клавіатура і т.ін.), але й виведення (запити на рисування). Події можуть генеруватися як регіонами, які процеси помістили в простір подій, так і самим мікроядром Photon. Визначеними є такі типи подій: натиск клавіш, натиск пера й кнопок мишки, переміщення пера й мишки, пересікання границь регіону, події експозиції й перекриття, події рисування, події перетаскування (drag).

Додатки можуть чекати настання подій (і при цьому блокуватися) або отримувати асинхронні повідомлення про прихід події.

Список прямокутників, прикріплених до події, може визначати одну або більше прямокутних областей чи "первісну точку" – єдиний прямокутник, у якого координати верхнього лівого й нижнього правого кутів збігаються.

При пересіканні подією непрозорого регіону, прямокутник регіону "вирізається" зі списку прямокутників події так, що список описує тепер лише видиму частину події.

Краще всього ілюструє таке відсікання те, як змінюється список прямокутників події рисування при її проходженні крізь різні регіони. Коли подія рисування генерується, список прямокутників вміщує єдиний прямокутник, який описує регіон, народжений подією.

Якщо подія проходить через регіон, який відсікає, наприклад, верхній лівий кут події рисування, то список прямокутників модифікується і буде вміщувати вже два прямокутники, які визначають область, яка підлягає рисуванню. Ці результуючі прямокутники називають "плитки" (tiles).

Подібно до цього, кожен раз при пересіканні подією рисування непрозорого регіону, список прямокутників буде модифікуватися таким чином, щоб описувати область, яка залишається видимою після "вирізання" непрозорого регіону. Коли, нарешті, подія рисування досягне графічного драйвера, то список прямокутників буде точно описувати тільки її видиму частину.

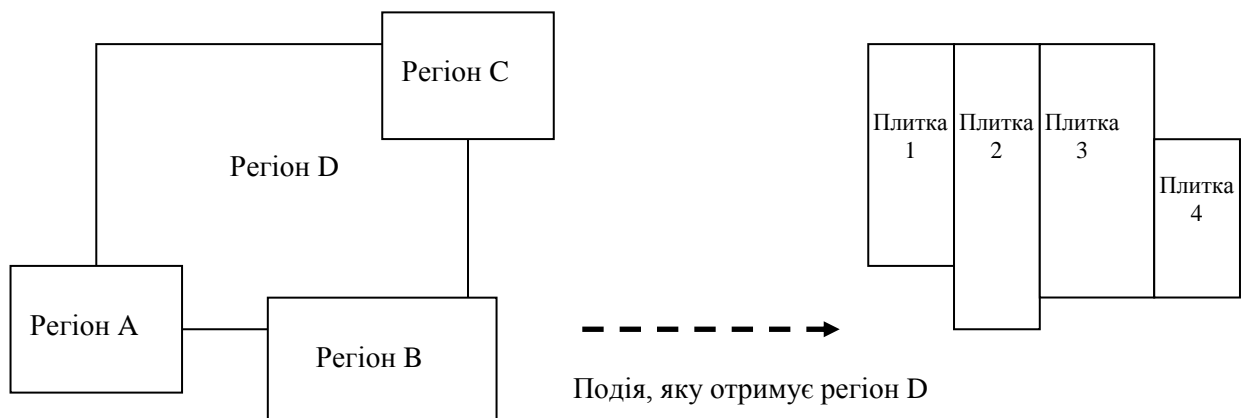


Рис.2. Формування списку прямокутників

Непрозорі для події рисування регіони вирізаються, в результаті чого отримується область, яка складається з прямокутних "плиток".

В тому випадку, якщо подія рисування цілком відсікається при пересіканні з регіоном, вона припиняє своє існування. Цей механізм "непрозорих" вікон, що змінює список прямокутників події рисування, забезпечує правильне відсікання подій рисування під час їх просування від первісного регіону (та зв'язаного з ним процесу) до користувача.

### **1.5. Графічні драйвери**

Графічні драйвери реалізовані як процеси, які розміщують регіон на передньому плані простору подій. Регіон графічного драйвера є чутливим до подій рисування, які виходять з простору подій. Графічний драйвер отримує події рисування, коли вони пересікають його регіон. Можна представити собі, що цей регіон є покритим "фосфором", який світиться при попаданні "фотонів".

Оскільки API рисування Photon накопичують запити рисування в пакети, які відправляються як одна подія рисування, то кожна подія рисування, яку отримує драйвер, вміщує список графічних примітивів, які підлягають рисуванню. До моменту пересікання подією рисування регіону драйвера, список прямокутників буде вміщувати також "список відсікань", який описує, які саме частини списку рисування повинні відображатися на дисплеї. Робота драйвера полягає в тому, щоб перетворити результуючий список у візуальне зображення на графічному обладнанні.

Одна з переваг використання списку прямокутників всередині події полягає в тому, що кожна подія, яка передається драйверу, представляє собою фактично "пакет" запитів. Із вдосконаленням графічного обладнання все більше й більше такої "пакетної" роботи може передаватися безпосередньо обладнанню. Багато відеоадаптерів вже підтримують апаратно одну область відсікання, а деякі підтримують й декілька областей.

Використання механізму QNX IPC для передачі запитів рисування від додатків до графічного драйвера не є збитковим, оскільки відповідні тести показують, що в цьому випадку продуктивність системи не є гіршою, ніж у випадку, коли додатки виконують прямі виклики драйвера. Однією з причин цього є те, що при використанні подій численні запити рисування групуються, що зменшує кількість повідомлень, які відправляються, у порівнянні з кількістю прямих викликів драйвера.

З того, що графічний драйвер просто розміщує регіон в просторі подій Photon, природно випливає, що одночасно можуть бути завантажені декілька графічних драйверів для декількох відео адаптерів; при цьому кожен драйвер буде мати свій, чутливий до подій рисування, регіон.

Ці регіони можуть бути розташовані поруч або перекривати один одного довільним чином. Оскільки Photon успадковує від QNX мережеву прозорість, то додатки або драйвери Photon можуть виконуватися на довільному вузлі мережі, дозволяючи, таким чином, додатковим графічним драйверам розширювати графічний простір Photon за рахунок фізичних дисплеїв інших комп'ютерів мережі. За рахунок перекриття регіонів графічних драйверів, події рисування можуть дублюватися на декількох екранах.



Завдяки зазначеним властивостям Photon можливими є такі ситуації. Наприклад, на заводі оператор з портативним комп'ютером, який має бездротове підключення до мережі, може підійти до робочої станції й "перенести" панель керування з її монітора на екран портативного комп'ютера, а потім перейти в цех і виконати керування. В інших додатках вбудована система без графічного інтерфейсу користувача може проєциувати дисплей на довільний із вузлів мережі. Крім того, стає можливим колективний режим роботи - декілька операторів, які знаходяться за своїми комп'ютерами, можуть одночасно бачити одні й ті ж вікна та працювати з одним й тим самим додатком. З точки зору додатку, це виглядає як один єдиний графічний простір. З точки зору користувача, це виглядає як група поєднаних комп'ютерів, де можна перетаскувати вікна з одного фізичного екрану на інший.

### **1.6. Кольорова модель**

Для представлення кольорів використовується 24-бітна RGB-модель (по 8 біт для червоного, зеленого й синього), що забезпечує 16,777,216 кольорів. Залежно від типу обладнання, драйвер або безпосередньо відображає 24-бітний колір, або використовує різні варіанти змішування кольорів, щоб відобразити потрібний колір на обладнанні, яке підтримує меншу кількість кольорів.

Оскільки графічні драйвери використовують апаратно-незалежне представлення кольорів, то додатки можуть працювати без змін на різному обладнанні незалежно від того, яку кольорову модель воно підтримує. Це дозволяє "перетаскувати" додатки з одного монітора на інший, не замислюючись над тим, яка кольорова модель апаратно реалізована в кожному конкретному випадку.

### **1.7. Шрифти**

Photon підтримує растрові та масштабовані шрифти. Останні можуть масштабуватися практично з довільним розміром точки й використовувати технологію сгладжування (16 відтінків) для чіткого й ясного відображення на екрані з довільною роздільною здатністю.

Масштабовані шрифти в Photon підтримуються швидкодіючим сервером шрифтів, який завантажує опис шрифтів, які зберігаються в стисненому вигляді в файлах \*.pfr (Portable Font Resource - ресурси портативних шрифтів), й потім приводить вигляд символів у відповідність з довільним розміром точки й роздільною здатністю. Зауважимо, що формат PFR забезпечує більше ніж в два рази кращий стиск у порівнянні з PostScript шрифтами.

Основний латинський (Core Latin) набір шрифтів Photon (latin1.pfr), який охоплює два набори символів стандарту Unicode, Basic Latin (U+0000 - U+007F) й Latin-1 Supplement (U+0080 - U+00FF), включає такі масштабовані шрифти:

- Dutch;
- Dutch Bold;
- Dutch Italic;
- Dutch Bold Italic;
- Swiss;
- Swiss Bold;
- Swiss Italic;
- Swiss Bold Italic;
- Courier;
- Courier Bold;
- Courier Italic;
- Courier Bold Italic.

Розширений латинський (Extended Latin) набір ([latinx.pfr](#)) охоплює набори символів Unicode Latin Extended-A (U+0100 – U+017F) й Latin Extended-B (U+0180 - U+021F) та включає такі шрифти:

- Dutch;
- Dutch Bold;
- Dutch Italic (генерується алгоритмічно);
- Dutch Bold Italic (генерується алгоритмічно);
- Swiss;
- Swiss Bold;
- Swiss Italic (генерується алгоритмічно);
- Swiss Bold Italic (генерується алгоритмічно).

За допомогою основного латинського набору ([latin1.pfr](#)) можна підтримувати множину мов, включаючи такі мови:

Датську;	Фламандську;	Ісландську;	Норвежську;
Голландську;	Французьку;	Індонезійську;	Португальську;
Англійську;	Німецьку;	Ірландську;	Іспанську;
Фінську;	Гавайську;	Італійську;	Суахілі;
			Шведську.

Розширений набір ([latinx.pfr](#)) дозволяє додатково підтримувати:

Африканську;	Есперанто;	Литовську;	Турецьку;
Баскську;	Естонську;	Мальтійську;	Валлійську.
Каталонську;	Гренландську;	Польську;	
Хорватську;	Угорську;	Румунську;	
Чеську;	Латиську;	Словацьку;	

Для Photon пропонуються декілька додаткових пакетів для підтримки національних мов:

- Японської;
- Китайської;
- Корейської;
- Кирилиці.

Зауважимо також, що Photon був розроблений з урахуванням підтримки національних символів за стандартом Unicode (ISO/IEC 10646), і тому надає можливість створювати додатки, які підтримують основні світові мови. Як відомо, Unicode є заснованим на наборі ASCII-символів і використовує 16-бітне кодування для повної підтримки багатомовного тексту. Одним із варіантів є 8-бітна форма кодування UTF-8, яка визначає використання символів Unicode в 8-бітному середовищі UNIX. Основні характеристики UTF-8 такі:

- Unicode-символи від U+0000 до U+007E (набір ASCII) відображаються в UTF-8-байти від 00 до 7E (ASCII-значення);
- ASCII-значення не зустрічаються іншим чином в UTF-8, що забезпечує повну сумісність з файловими системами, які аналізують ASCII-байти;
- UTF-8 спрощує перетворення в Unicode-текст та з нього;
- Перший байт вказує кількість байт в послідовності, забезпечуючи ефективний розбір;
- Можна легко знайти начало символу з довільного місця в потоці байт, оскільки для цього необхідно перебрати не більше чотирьох байт, а початковий байт легко визначити. Наприклад: `isInitialByte = ((byte & 0xC0) != 0x80);`
- UTF-8 є досить компактним за кількістю байт, які використовуються для кодування.

Відповідна системна бібліотека включає такий ряд функцій для перетворень:

`mblen()` – довжина багатобайтового рядка в символах;  
`mbtowc()` – перетворити багатобайтовий символ в двохбайтовий символ;  
`mbstowcs()` – перетворити багатобайтовий рядок в двохбайтовий рядок;  
`wctomb()` – перетворити двохбайтовий символ в його багатобайтове

представлення;

`wcstombs()` – перетворити рядок двохбайтових символів в багатобайтовий рядок.

Додатково до зазначених функцій можна також використати власну бібліотеку Photon (функції `PxTranslate`) для виконання різних перетворень наборів символів в/із UTF-8.

### 1.8. Підтримка анімації та друку

Photon забезпечує немерехтливу анімацію через спеціальний віджет-контейнер з "подвійним буфером" (`PtDBContainer`), який створює спеціальний контекст в пам'яті для рисування зображень. Віджет `PtDBContainer` використовує блок поділюваної пам'яті, який є достатнім для збереження зображення відповідного розміру.

Photon передбачає вбудовану підтримку друку з виведенням на різні пристрої, включаючи: файли бітових карт, PostScript, Hewlett-Packard PCL, Epson ESC/P2, Canon, Lexmark. Photon також має віджет-діалог для вибору принтера з додатків.

### 1.9. Менеджер вікон Photon

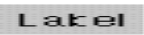
Додавання менеджера вікон PWM перетворює Photon в повнофункціональний графічний інтерфейс. Менеджер вікон не є обов'язковим і може бути відсутнім в більшості вбудованих систем. Менеджер вікон дозволяє маніпулювати вікнами додатків, змінюючи їх розмір, пересуваючи та мінімізуючи.


Менеджер вікон використовує концепцію фільтрації подій. Він розташовує додаткові регіони за регіонами додатків, на яких "нарисовані" елементи керування вікнами. Оскільки вид та поведінка інтерфейсу визначаються менеджером вікон, то можна реалізувати різні види інтерфейсів.

## 2. Бібліотека віджетів

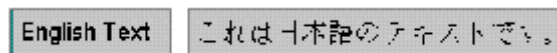
Photon має бібліотеку компонентів, які називають віджетами, що спроможні автоматично керувати своєю поведінкою без їх безпосереднього програмування. Будь-який додаток може бути швидко зібраний з віджетів з наступною прив'язкою C-коду до відповідних callback-функцій (функцій зворотного виклику) віджетів. Середовище Photon Application Builder (PhAB) є частиною системи розробки Photon і підтримує широку палітру віджетів в візуальному середовищі розробки. Photon пропонує широкий спектр віджетів, а саме: базові віджети (наприклад, кнопка), віджети-контейнери (наприклад, вікно), складні віджети (наприклад, HTML-віджет).

### 2.1. Базові віджети

Віджет «Ярлик» (PtLabel) має такий графічний вигляд :  і використовується для відображення текстової інформації. Віджет PtLabel є надкласовим для всіх текстових віджетів та забезпечує настройку різних атрибутів (наприклад, шрифт, спливаючі балони, кольори, бордюр, вирівнювання, поля и т.д.), які успадковуються всіма підкласами.

Віджет «Кнопка» (PtButton) має такий графічний вигляд:  та є невід'ємним компонентом будь-якої віконної оболонки. Звичайно цей віджет має опуклий вид, який при натисканні змінюється на утоплений, візуально відображаючи вибір кнопки. Додатково до візуального відображення стану при виборі кнопки автоматично викликається визначена додатком callback-функція.

Віджети «Текстове введення» (PtText, PtMultiText) мають такий вигляд:



Віджет PtText є простий однорядковий віджет, який звичайно застосовується у формах. Віджет PtMultiText є багаторядковий віджет, який забезпечує можливості редагування, автоматичної прокрутки та підтримує множину шрифтів.

Віджети «Кнопка з фіксацією» (PtToggleButton, PtOnOffButton) мають такий вигляд:



Ці віджети відображають один з двох можливих станів – включено або виключено, причому є два різних типи кнопок з фіксацією з різним зовнішнім виглядом. Віджети «Кнопка з фіксацією» використовуються для відображення або введення інформації про стан певної команди або дії.

Графічні віджети (PtArc, PtPixel, PtRectangle, PtLine, PtPolygon, PtEllipse, PtBezier, PtGrid) дозволяють формувати графічні зображення від простих ліній та прямокутників до складних багатосегментних кривих Без'є (Bézier) (табл. 2).

Таблиця 2

#### Статичні графічні об'єкти

Компонент	Призначення
PtPixel	Побудова множини пікселів
PtArc	<b>Побудова секторів, дуг, кіл й еліпсів</b>
PtRect	Побудова прямокутників та прямокутників із закругленими краями
PtLine	Побудова прямих
PtPolygon	Побудова поліліній та багатокутників
PtEllipse	Побудова окружностей и еліпсів
PtBezier	Побудова кривих Без'є

Для того, щоб встановити потрібний графічний об'єкт на форму, потрібно клацнути на ньому в системному вікні Widgets, а потім розтягти на формі до потрібного розміру. Розміри й положення об'єкта можна змінювати мишкою або за допомогою того ж системного вікна Widgets. У кожного з графічних компонентів є набір властивостей, якими можна змінювати зовнішній вигляд об'єкта (табл. 3).


Таблиця 3


#### Атрибути статичних графічних об'єктів

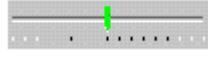
Віджет та його властивості	Призначення властивості
1	2
<b><u>PtPixel</u></b>	
Color:Pen	Колір точок
Color:Fill	Колір області побудови
<b><u>PtArc</u></b>	
Start Angle	Початковий кут (в десятих градуса, тобто для 180 градусів = 1800)
End Angle	Початковий кут (в десятих градуса)
Line Width	Товщина лінії
Color:Pen	Колір дуги
Color:Fill	Колір області побудови
Inside Color	Колір внутрішньої області
<b><u>PtRect</u></b>	
Roundness	Ступінь закругленості країв (пікселів)
Line Width	Товщина лінії

1	2
Color:Pen	Колір лінії країв
Color:Fill	Колір області побудови
Inside Color	Колір внутрішньої області
<b>PtLine</b>	
Line Width	Товщина лінії
Color:Pen	Колір точок
Color:Fill	Колір області побудови
<b>PtPolygon</b>	
Line Width	Товщина лінії
Color:Pen	Колір лінії
Color:Fill	Колір області побудови
Inside Color	Колір внутрішньої області
<b>PtEllipse</b>	
Line Width	Товщина лінії
Color:Pen	Колір лінії краю
Color:Fill	Колір області побудови
Inside Color	Колір внутрішньої області
<b>PtBezier</b>	
Line Width	Товщина лінії
Color:Pen	Колір лінії краю
Color:Fill	Колір області побудови
Inside Color	Колір внутрішньої області

Для побудови компонентів PtBezier, PtPolygon, PtPixel в PhAB є спеціальний редактор вузлів, який дозволяє додати потрібне число точок.

Віджет «Смуга прокрутки» (PtScrollbar) має вигляд:  і використовується для прокрутки зображення у видимій області. «Смуга прокрутки» також використовується у складі інших віджетів (наприклад, PtList, PtScrollArea) для забезпечення прокрутки.

Віджет «Роздільник» (PtSeparator) має вигляд:  і використовується для поділу двох або більше областей, що надає кращий зовнішній вид. «Роздільник» може бути настроєний відповідно до різних стилів та виглядів.

Віджет «Движок» (PtSlider) має вигляд:  та відрізняється від «смуги прокрутки» тим, що смуга прокрутки визначає інтервал, тоді як «движок» задає єдине значення. Віджет «Движок» має достатньо великий список атрибутів.

Віджет «Таймер» (PtTimer) істотно спрощує використання таймера. Цей віджет не має графічного образу, а лише визначає callback-функцію, яка викликається кожен раз при спрацьовуванні таймера. Додаток може установлювати значення таймера та, за вибором, інтервал повторення.

Віджети «Графічне зображення» (PtBitmap, PtLabel, PtButton) дозволяють імпортувати зображення та показувати їх усередині віджетів. Багато віджетів Photon безпосередньо підтримують таке відображення графіки, а найбільш

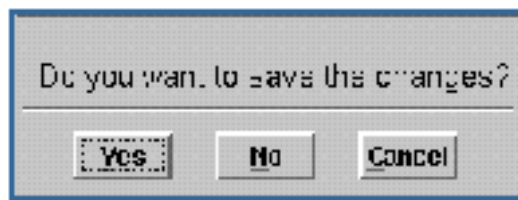
використовуваними є `PtButton` для створення панелей кнопок та `PtLabel` для показу зображень.

Віджет «Індикатор ходу процесу» (`RtProgress`) має вигляд :



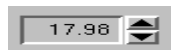
і використовується для демонстрації ходу виконання тривалої операції (наприклад, завантаження файлу). Індикатор ходу процесу може бути горизонтальним або вертикальним та має багато атрибутів, які можна налаштувати.

Віджет «Повідомлення» (`PtMessage`) має вигляд:



і реалізує зручний діалог з показом повідомлення та трьома кнопками для відповіді. Зауважимо, що функція виклику модального діалогу (`PtAskQuestion()`) заснована на даному віджеті.

Числові віджети (`PtNumericInteger`, `PtNumericFloat`) мають вигляд:

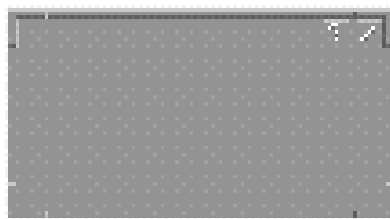


Віджет `PtNumericInteger` дозволяє задавати цілочислові значення в межах між установленими мінімальною й максимальною величинами. Віджет `PtNumericFloat` дозволяє увести число з плаваючою точкою.

Віджет `PtUpDown` (стрілки вгору/вниз) дозволяє збільшувати або зменшувати число на задану величину.

## 2.2. Віджети-контейнери

Віджети «Вікно» та «Піктограма» (`PtWindow`, `PtIcon`) мають вигляд:




Вікна є основними контейнерами для додатків. Основні компоненти графічного інтерфейсу (лінійки меню, лінійки інструментів і т.ін.) з'являються з віджетом «Вікно». Цей віджет автоматично виконує всі необхідні взаємодії з Менеджером вікон Photon (`PWM` – Photon Window Manager) і потребує лише завдання потрібної функціональності.

Віджет «Піктограма» тісно пов'язаний з вікном і показується в папках Photon Desktop Manager та на панелі задач `PWM`.

Віджет «Панель» (`PtPane`) має вигляд:



і є простим віджет-контейнером, який використовується для розміщення інших віджетів. Незважаючи на те, що він є батьківським віджетом, віджет «панель» ніяким чином не керує дочірніми віджетами. Панелі дуже зручні для побудови форм, які звичайно зустрічаються у вікнах діалогу.

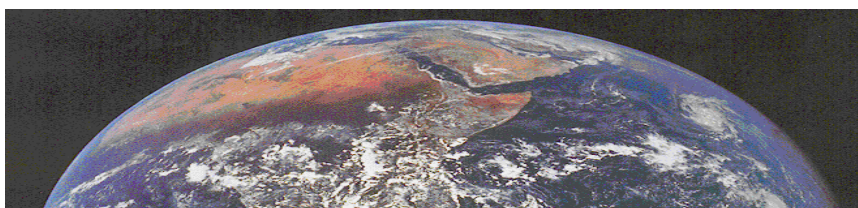
Віджет «Група» (PtGroup) () керує геометрією всіх своїх дочірніх віджетів. Він може вирівнювати віджети горизонтально, вертикально або у вигляді матриці. Група може бути прив'язана до сторони довільного іншого контейнера (наприклад, вікна) таким чином, щоб група автоматично змінювала розмір при зміні розміру вікна. Віджет «Група» також має атрибути, які дозволяють задати необхідність розтягування дочірніх віджетів при збільшенні розмірів групи.

Віджет «Область прокрутки» (PtScrollArea) має вигляд:



та забезпечує "вікно" перегляду умісту контейнера потенційно більшого розміру. Якщо помістити певну кількість віджетів усередину області прокрутки, то він автоматично створить смуги прокрутки у випадку, коли віджети будуть виходити за межі видимої області. Області прокрутки можуть бути використані для створення вікна перегляду текстових файлів, текстових редакторів, перегляду списків і ін. Для швидкої прокрутки дочірніх віджетів область прокрутки використовує бліттер (апаратний засіб копіювання фрагментів зображення між різними частинами буфера) за умови, що він підтримується графічним драйвером.

Віджет Фон (PtBkgd) може мати вигляд:

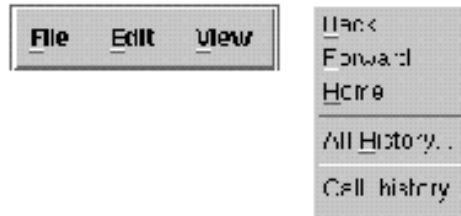




та дозволяє створювати певний фон, починаючи від простого переходу кольорів до симетрично розташованих текстур.

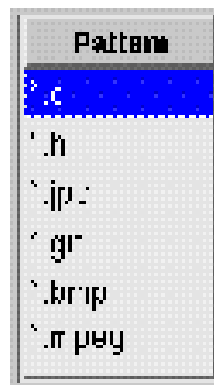
### 2.3. Складні віджети

Віджети Меню (PtMenu, PtMenuBar, PtMenuButton) мають вигляд:

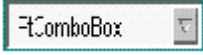


та дозволяють створити меню. Віджет PtMenuBar спрощує створення стандартної лінійки меню. Віджет PtMenuBar дозволяє відображати спливаюче меню, обробляти натискання-переміщення-відпускання (мишки), вказівку й натискання, введення з клавіатури та вибір пунктів меню. Віджет PtMenuButton використовується для створення окремих пунктів меню.

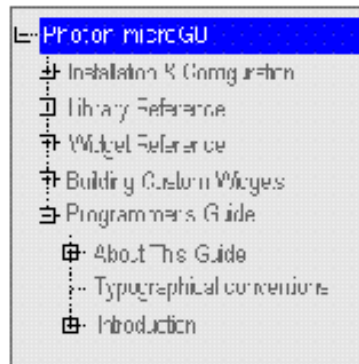
Віджет «Список» (PtList) може мати вигляд:



і застосовується для керування списком елементів. Він має багато різних режимів вибору, включаючи одиничний вибір, множинний вибір та вибір діапазону. Віджет «Список» підтримує багатостолбцеві списки при використанні віджета PtDivider.

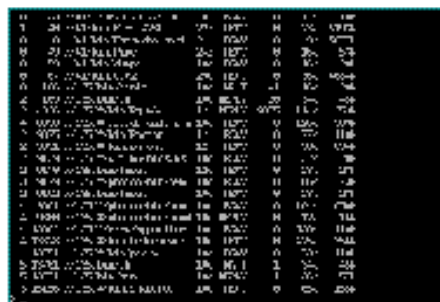
Віджет «Список, що розгортається» (PtComboBox) має вигляд:  і сполучає віджет PtText (для введення тексту) з кнопкою для відображення віджета PtList. Під час вибору елемента списку, віджет «Текст» автоматично оновлюється відповідно до поточного вибору. Віджет «Список, що розгортається» є дуже корисним для відображення списку в обмеженому просторі. Діалоги та контейнери займають значно менше місця на екрані, що дуже важливо для вбудованих додатків.

Віджет «Дерево» (PtTree) має вигляд:



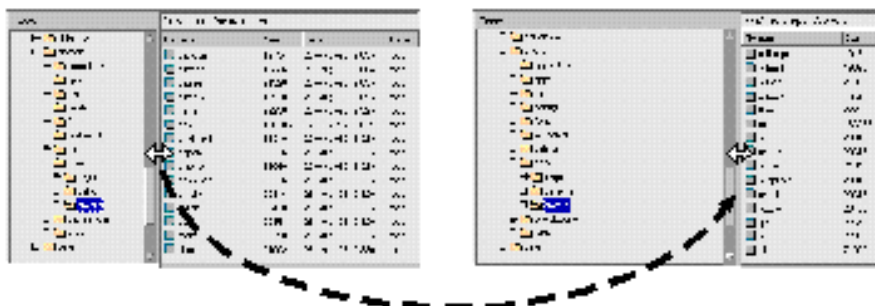
та нагадує віджет «Список», оскільки вони мають спільних попередників. Головна відміна полягає в тому, що віджет «Дерево» показує елементи у вигляді ієрархії. Елементи, які називають гілками, можуть бути розгорнуті або зжаті; може бути створена будь-яка кількість гілок. Для кожної гілки можна визначити своє унікальне графічне зображення. До числа додатків Photon, які використовують дерева, входять Файл-Менеджер (показ каталогу), PhAB (ієрархія віджетів), vsin (список процесів) та багато інших.

Віджети «Термінал» (PtTty, PtTerminal) мають вигляд:



та дозволяють помістити текстову консоль у свій додаток. Віджет «Термінал» створює текстовий термінал та керує ним. Більше того, він забезпечує повну функціональність "cut-and-paste" та швидкий виклик довідки шляхом виділення тексту усередині віджета.

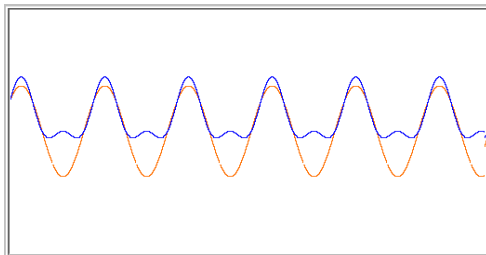
Віджет «Велитель» (PtDivider) має вигляд:



та здійснює керування дочірніми віджетами наступним чином. Якщо помістити два або більше віджетів усередину віджета PtDivider, то він автоматично створює невеликі роздільники між дочірніми віджетами. Пересуваючи ці роздільники, можна змінити розміри дочірніх віджетів. Це є дуже зручним для створення списків зі

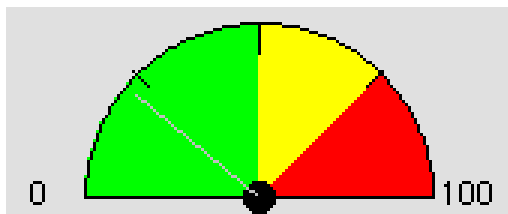
стовпцями змінюваної ширини. Фактично, якщо помістити віджет PtDivider усередину PtList, то це автоматично перетворить простий список в список з множинними стовпцями змінюваної ширини.

Віджет «Тренд» (RtTrend) має вигляд:



і призначений для відображення графічних трендів стану процесу. Віджет RtTrend підтримує відображення декількох трендів одночасно.

Віджет «Вимірювальний прибор» (RtMeter) має вигляд:



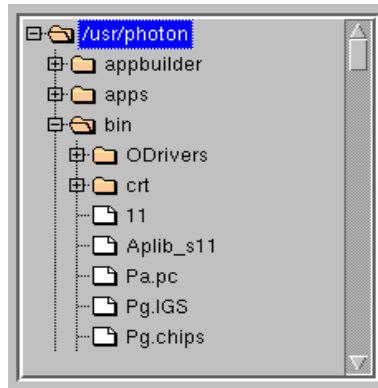
тобто вигляд півкруга з рисками, що відзначають 1/3, 1/2 і 2/3 довжини дуги. Стрілка може переміщатися за допомогою мишки, або клавіатури, або програмно. Однократне натискання кнопки мишки переміщає стрілку в поточну позицію курсору; при натисканні й наступному переміщенні мишки ("drag") стрілка прямує за курсором.

Віджет «Діалог вибору шрифту» (PtFontSel) має вигляд:



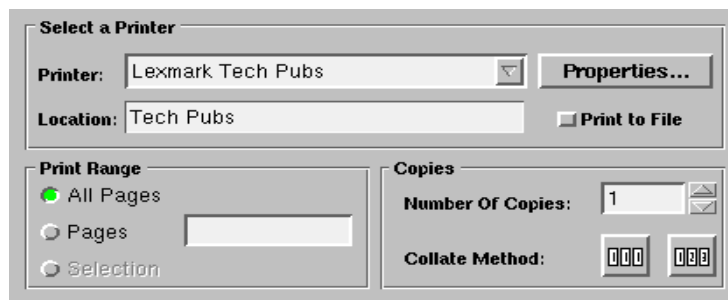
та вміє читати стандартні файли конфігурації шрифтів й показує список доступних шрифтів. Він дозволяє вибрати шрифт та стиль (жирний, курсив і т.і.), а також вказати на необхідність використання технології згладжування (anti-alias).

Віджет «Вибір файлу» (PtFileSel) має вигляд:



та дозволяє відображати деревоподібну ієрархію файлів, каталогів або довільних елементів. За допомогою цього віджету можна переглядати структуру файлової системи та вибирати потрібний файл чи каталог.

Віджет «Діалог настройки друку» (PtPrintSel) має вигляд:



та дозволяє обрати принтер і провести необхідну настройку параметрів друку, причому можна задавати діапазон сторінок для виведення на друк й кількість копій.

Віджет «HTML» (PtHtml) полегшує створення власного засобу перегляду документів за форматом HTML. Віджет сам виконує форматування стандартного HTML-файлу та автоматично завантажує картинки. Він обробляє прокрутку, зміну розміру та практично всі інші потрібні функції.

## 2.5. Створення нових віджетів

Якщо стандартних віджетів Photon недостатньо, то можна легко створити свої власні нові віджети. До складу середовища розробки Photon входить повна документація та приклади коду для створення власних віджетів. Можна створювати підкласи існуючих віджетів, щоб забезпечити спадкування їх функціональності, або створити власне дерево віджетів.

## 3. Приклад розробки простого графічного додатку в середовищі PhAB

Для завантаження та запуску середовища PhAB потрібно відкрити термінал (консоль) на подати команду `ab`. Після цього буде відкрито головне вікно середовища PhAB. Якщо зазначене вікно не відкрилося, це може вказувати на відсутність системного програмного забезпечення для PhAB.

Після відкриття головного вікна PhAB можна створювати графічний додаток. Для прикладу розглянемо створення графічного додатку у вигляді вікна плоскої форми, в якому будуть розташовані примітивні графічні об'єкти, причому

натискання лівої кнопки мишки на основному полі вікна призведе до зміни кольорів графічних об'єктів.

Для створення зазначеного додатку необхідно обрати в головному меню середовища PhAB наступні пункти: “File” – “New” – “Plain App”. В результаті цих дій отримаємо порожню форму вікна (рис.2) з іменем base за класом PtWindow.

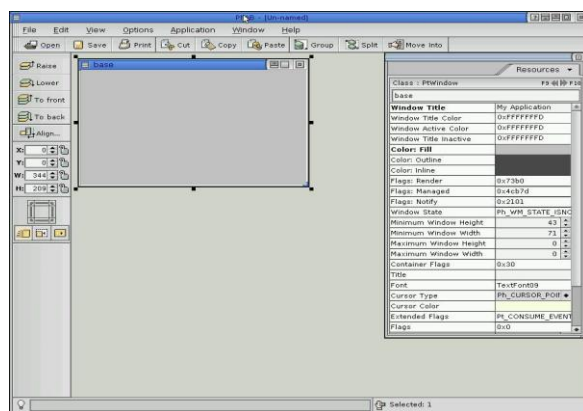


Рис. 2. Графічна оболонка PhAB з порожньою формою плоского вікна (ліворуч вгорі) та системним вікном PhAB, яке визначає властивості (ресурси) цієї форми (праворуч)

Далі відкриваємо системне вікно Widgets та за допомогою мишки створюємо (шляхом «перетаскування» потрібного віджета з системного вікна Widgets на основне поле форми графічного додатку) графічні об'єкти PtArc, PtPixel, PtRect, PtLine, PtPolygon, PtEllipse та PtBezier. Для зміни властивостей цих графічних об'єктів необхідно «вказати» мишкою на певний об'єкт та увести нове значення для обраної властивості у відповідне поле. Наприклад, для PtArc: Start Angle = 1800, Color:Pen = зелений, Inside Color = зелений; для PtPixel: Color:Pen = бордовий; для PtRect: Roundness = 6, Inside Color = синій; для PtLine: Line Width = 4; для PtPolygon: Inside Color = рожевий; для PtEllipse: Color:Pen = жовтий, Inside Color = жовтий. Після цього отримаємо результат, подібний до результату на рис.3, бо він залежить від того, як були розташовані елементи та від обраного числа точок для побудови певних графічних об'єктів.

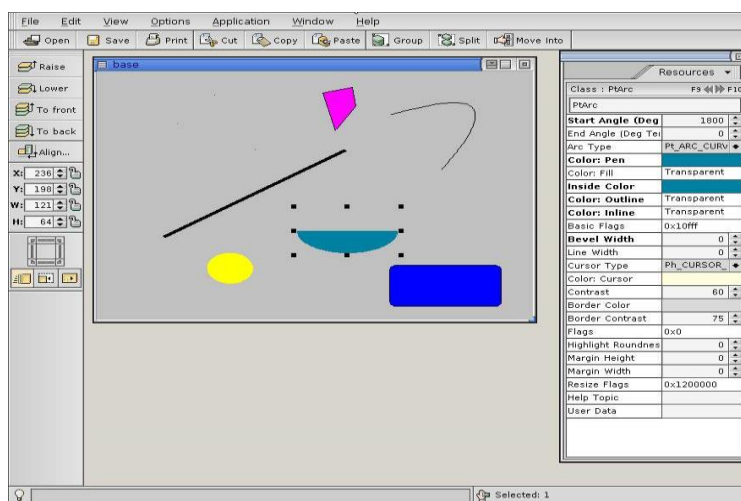


Рис. 3. Приклад з розташуванням графічних об'єктів на формі плоского вікна

Для того, щоб забезпечити зміну кольорів графічних об'єктів шляхом натискання лівої кнопки мишки на основному полі вікна, необхідно приєднати відповідний код, за допомогою якого виконується зазначена зміна кольорів, до коду callback-функції основного поля вікна. Це можна зробити за допомогою редактора callback-функцій. Але перед цим необхідно виконати зміну назв всіх екземплярів графічних об'єктів на інші, які не повинні збігатися з іменами відповідних класів; ці назви можна переглянути в системному вікні Widgets шляхом вказівки курсором мишки на відповідний графічний об'єкт та натисканням лівої кнопки мишки. Наприклад, для графічного об'єкту PtArc можна встановити назву PtArc1. Крім того, є необхідною також зміна назви основного вікна з іменем "base" на інше ім'я, наприклад, "base1". Після виконання зазначених змін (особливо, для тих об'єктів, для яких будуть виконуватися зміни коду їх callback-функцій) необхідно зберегти розроблений графічний додаток шляхом вибору пунктів головного меню "File" – "Save as" середовища PhAB.

Формування коду callback-функцій виконується за таким алгоритмом: 1) обрати віджет; 2) включити Callbacks control panel; 3) обрати тип callback зі списку (наприклад, Activate, що означає натискання та відпускання лівої кнопки мишки); 4) для додавання нової callback-функції у вікні "Callbacks control panel" обрати "New", після чого на екрані з'явиться вікно текстового редактора з автоматично генерованим кодом callback-функції, яке рекомендується зачинити і подальше редагування тексту програмного коду callback-функції виконувати пізніше за допомогою текстового редактору `red`; 5) для нової callback-функції обрати тип "Code Types"; 6) заповнити поле "Link to Callback/Module info"; 6) натиснути кнопку "Apply" для застосування уведених змін.

Редагування коду callback-функцій краще виконувати за допомогою текстового редактору `red`. Для цього слід знайти відповідний файл з текстом callback-функції та відкрити його за допомогою зазначеного редактора. Якщо callback-функція призначена для зміни, наприклад, кольору об'єкту PtArc1, то всередину тексту callback-функції перед оператором `return(Pt_CONTINUE)` необхідно набрати текст такого оператора: `PtSetResource(ABW_PtArc1, Pt_ARG_FILL_COLOR, Pg_RED, 0)`. В цьому випадку колір об'єкту PtArc1 зміниться на червоний у разі натискання та відпускання лівої кнопки мишки на основному полі вікна. Аналогічно можна встановлювати чи змінювати інші "ресурси" цього об'єкту та інших об'єктів. Слід зауважити, що на відміну від середовища програмування C++Builder, де застосовується ім'я інстанції програмного компонента, в програмному середовищі PhAB до кожної інстанції віджета необхідно звертатися тільки через її маніфест (вказівник на інстанцію віджета), для чого перед іменем об'єкту необхідно приписувати префікс "ABW\_".

Після виконання дій, що пов'язані з редагуванням коду callback-функцій, необхідно виконати компіляцію розробленого графічного додатку. Для цього слід в головному меню обрати пункти "Application" – "Build + Run". Після цього з'явиться вікно компіляції (рис. 4).

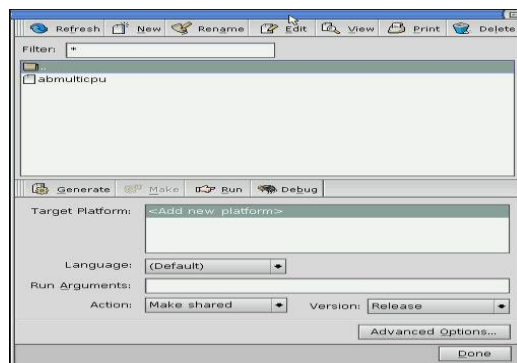


Рис. 4. Вікно компіляції

Спочатку в даному вікні треба натиснути кнопку "Generate" (щоб створити шаблони відповідних функцій). Після появи вікна вибору програмної платформи необхідно обрати GCC, і тоді будуть створені обробники подій та шаблони для коду (на цьому етапі програміст може внести корективи в код програми). Далі слід натиснути кнопку "Make" для виконання компіляції коду (рис. 5).

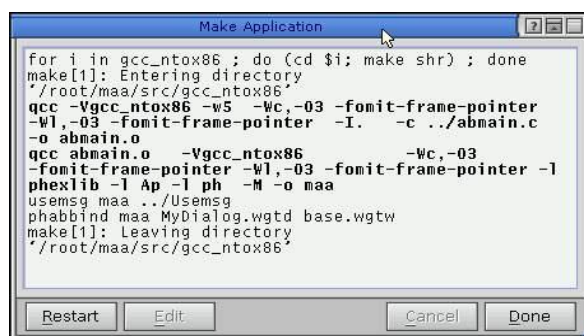


Рис. 5. Результат компіляції коду

Якщо компіляція коду пройшла без помилок, то далі можна запустити графічний додаток на виконання шляхом натиску на кнопку "Run".

Розроблений графічний додаток можна запустити на виконання з консолі терміналу шляхом указівки його імені.

### Контрольні запитання

1. Яке призначення мікроядра Photon?
2. Що означає поняття "віджет"?
3. Наведіть приклади віджетів.
4. Що означає поняття "callback-функція"?
5. Що означає команда ab?
6. Як виконати зміну властивості (ресурсу) віджету?
7. Як змінити назву інстанції віджету?
8. Як виконати компіляцію графічного додатку?
9. В якій папці знаходиться програмний код змінюваної callback-функції?
10. Дайте перелік основних віджетів, які можна використати для керування діями графічного додатку.
11. Чи можна створити власні віджети?
12. Чи можна завантажити на виконання графічний додаток поза середовищем PhAB?

## Завдання до лабораторної роботи

### Завдання 1

Розглянути властивості визначених (за варіантом завдання) віджетів та на їх основі створити простий графічний додаток, в якому продемонструвати виклики callback-функцій. Відповідність варіантів та набору віджетів наведена в табл. 4.

Таблиця 4

Варіанти до завдання 1

Номер варіанту	Віджети
1	PtMessage, PtSlider, PtButton
2	PtBitmap, PtComboBox, PtMenuButton
3	PtMenu, PtArc, PtSeparator
4	PtContainer, PtEllipse, PtText
5	PtPane, PtPolygon, PtList
6	PtGauge, PtToggleButton, PtGroup
7	PtBkgnd, PtPixel, PtMultiText
8	PtScrollArea, PtRect, PtOnOffButton
9	PtPane, PtLine, PtScrollBar
10	PtWindow, PtSeparator, PtMenu
11	PtMessage, PtToggleButton, PtLine
12	PtMenuButton, PtGroup, PtBkgnd
13	PtComboBox, PtRect, PtGauge
14	PtBitmap, PtSeparator, PtText
15	PtTree, PtToggleButton, PtWindow

### Завдання 2

На основі віджету реального часу «Тренд» розробити графічний додаток "Цифровий осцилограф", долучаючи до розробки інші потрібні віджети. Роботу даного графічного додатку продемонструвати на прикладі певних цифрових даних, які моделюють, наприклад, синусоїдальний сигнал у адитивній суміші з вимірювальним шумом. Визначити час, який витрачається на моделювання одного блоку даних, та час, який витрачається на відображення цього блоку даних на екрані, а також час, який витрачається на один повний цикл роботи (від початку моделювання до початку наступного моделювання). Розмір блоку даних визначати так: «розмір\_блоку\_даних=номер\_варіанту\_завдання\*100» цифрових відліків за типом short int.

### Завдання 3

Розробити графічний інтерфейс фрагменту системи реального часу відповідно до визначеного варіанту завдання 3 лабораторної роботи 1.



## Література

1. Сергей Зиль. Операционная система реального времени QNX от теории к практике. 2-е издание. Санкт-Петербург, “БХВ-Петербург”, 2004 – 191 с.
2. Сергей Зиль. QNX momentics основы применения. Санкт-Петербург, “БХВ-Петербург”, 2005 – 253 с.
3. Роберт Кртен. Введение в QNX Neutrino. Руководство для разработчиков приложений реального времени (2-е издание). Санкт-Петербург, “БХВ-Петербург”, 2011 – 368 с.

## Зміст

	Стр.
<u>Лаботаторна робота № 1. Основи роботи і налаштування ОСРЧ QNX .....</u>	3
<u>Лаботаторна робота № 2. Ознайомлення з інтегрованим середовищем розробки QNX та написання у ньому елементарної програми .....</u>	12
<u>Лаботаторна робота № 3. Побудова власного завантажувального образу QNX .....</u>	23
<u>Лаботаторна робота № 4. Ознайомлення з графічним середовищем розробки Photon Application Builder та набуття навичок для створення найпростіших програм .....</u>	33
<u>Лаботаторна робота № 5. Проектування систем реального часу на базі операційної системи QNX-6 .....</u>	46
<u>Лаботаторна робота № 6. Мікроядро Photon та інтегроване середовище Application Builder .....</u>	68
<u>Література .....</u>	89

## This image shows a single sheet of white paper with horizontal ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.

Навчально-методична література

**Чихіра І.В.**

**Методичні вказівки**  
**для виконання лабораторних робіт з дисципліни**  
**«Програмування систем реального часу»**

Комп'ютерне макетування та верстка *А.П. Катрич*

Формат 60х90/16. Обл. вид. арк. 3,64. Тираж 10 прим. Зам. № 2926.

Тернопільський національний технічний університет імені Івана Пулюя.

46001, м. Тернопіль, вул. Руська, 56.

Свідоцтво суб'єкта видавничої справи ДК № 4226 від 08.12.11.